PROVISIÓN DE SERVICIOS DE PROCESADO DE IMÁGENES BASADOS EN ARQUITECTURAS ORIENTADAS A SERVICIOS

Bernardino Marcos Asensio
12/09/2013

E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Arquitecturas SOA en sistemas empotrados

TÍTULO: Provisión de servicios de procesado de imágenes basados en arquitecturas orientadas a servicios (SOA)

AUTOR: Bernardino Marcos Asensio

TUTOR: Vicente Hernández Díaz **Vº Bº.**

DEPARTAMENTO: DIATEL

Miembros del Tribunal Calificador:

PRESIDENTE:

VOCAL: Vicente Hernández Díaz

VOCAL SECRETARIO:

DIRECTOR:

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

Con este proyecto se pretende crear un procedimiento general para la implantación de aplicaciones de procesado de imágenes en cámaras de video IP y la distribución de dicha información mediante Arquitecturas Orientadas a Servicios (SOA).

El objetivo principal es crear una aplicación que se ejecute en una cámara de video IP y realice un procesado básico sobre las imágenes capturadas (detección de colores, formas y patrones) permitiendo distribuir el resultado del procesado mediante las arquitecturas SOA descritas en la especificación DPWS (Device Profile for Web Services).

El estudio se va a centrar principalmente en la transformación automática de código de procesado de imágenes escrito en Matlab (archivos .m) a un código C ANSI (archivos .c) que posteriormente se compilará para la arquitectura del procesador de la cámara (arquitectura CRIS, similar a la RISC pero con un conjunto reducido de instrucciones).

Resumen

Con este proyecto se pretende crear un procedimiento general para la implantación de aplicaciones de procesado de imágenes en cámaras de video IP y la distribución de dicha información mediante Arquitecturas Orientadas a Servicios (SOA).

El objetivo principal es crear una aplicación que se ejecute en una cámara de video IP y realice un procesado básico sobre las imágenes capturadas (detección de colores, formas y patrones) permitiendo distribuir el resultado del procesado mediante las arquitecturas SOA descritas en la especificación DPWS (Device Profile for Web Services).

El estudio se va a centrar principalmente en la transformación automática de código de procesado de imágenes escrito en Matlab (archivos .m) a un código C ANSI (archivos .c) que posteriormente se compilará para la arquitectura del procesador de la cámara (arquitectura CRIS, similar a la RISC pero con un conjunto reducido de instrucciones).

Abstract

This project aims to create a general procedure for the implementation of image processing applications in IP video cameras and the distribution of such information through Service Oriented Architectures (SOA).

The main goal is to create an application that runs on IP video camera and carry out a basic processing on the captured images (color detection, shapes and patterns) allowing to distribute the result of process by SOA architectures described in the DPWS specification (Device Profile for Web Services).

The study will focus primarily on the automated transform of image processing code written in Matlab files (. M) to ANSI C code files (. C) which is then compiled to the processor architecture of the camera (CRIS architecture , similar to the RISC but with a reduced instruction set).

Quiero agradecer a mi familia el esfuerzo y dedicación volcados sobre mí todos estos años. En especial a mi madre que, en su afán por darme siempre lo mejor, insistió en que cursara estudios universitarios ya que ella nunca tuvo opciones. Seguramente no estaría trabajando cómodamente en una oficina de no ser por ella. Descansa en paz.

ÍNDICE

ÍNDICE v
ÍNDICE DE FIGURAS vii
1 INTRODUCCIÓN 1
1.1 PLANTEAMIENTO 1
1.2 OBJETIVOS CONSEGUIDOS 3
2 DESCRIPCIÓN DEL MARCO DE TRABAJO 5
2.1 CÁMARAS IP 5
2.1.1 CÁMARAS DIGITALES 5
2.1.2 SISTEMAS DE VIGILANCIA IP 6
2.1.3 RED IP 7
2.1.4 SELECCIÓN DE UNA CÁMARA PARA PRUEBAS 8
2.1.5 ARQUITECTURA DE LA CÁMARA IP 9
2.1.6 FIRMWARE 10
2.1.7 LINUX EMBEBIDO 11
2.1.8 PLATAFORMA DE APLICACIONES DE CÁMARAS AXIS 13
2.2 PROBLEMAS ENCONTRADOS 15
3 PROCESAMIENTO DE IMÁGENES 17
3.1 INTRODUCCIÓN 17
3.2 DESARROLLO DE UN ALGORITMO 17
3.2.1 MATLAB 18
3.2.2 SIMULINK 30
3.3 GENERACIÓN DE CÓDIGO DE PROCESADO DE IMÁGENES 33
3.4 LIBRERIAS DE PROCESAMIENTO DE IMÁGENES 35
3.4.1 OPENCV 35
3.4.2 QDBMP 35
3.5 PROBLEMAS ENCONTRADOS 45
4 ARQUITECTURAS SOA 47
4.1 INTRODUCCIÓN 47
4.2 DPWS 48

4.3 FRAMEWORKS PARA DESARROLLO DPWS 51
4.4 PROCESO DE DESARROLLO 51
4.4.1 DEFINICIÓN DEL DISPOSITIVO 52
4.4.2 DEFINICIÓN DEL SERVICIO 55
4.4.3 IMPLEMENTACIÓN DEL DISPOSITIVO 57
4.4.4 EJECUCIÓN DEL DISPOSITIVO 58
4.4.5 MONITORIZACIÓN SOAP 59
4.5 PROBLEMAS ENCONTRADOS 60
5 MODELOS PROPUESTOS 61
5.1 DIFICULTADES TÉCNICAS 61
5.2 DETECTOR DE MOVIMIENTO 61
5.3 CONTADOR DE PERSONAS 65
5.4 SISTEMA DE CAPTURA DE CARAS 67
5.5 COMPARADOR DEL VOLUMEN DE PIXELES 71
6 CASO PRÁCTICO 73
6.1 USO PRÁCTICO DEL COMPARADOR DE VOLÚMENES 73
6.1.1 CONTROL DE LLENADO DE RECIPIENTES 73
6.1.2 MOTIVOS DE LA ELECCIÓN 74
6.2 MODELO DEL SISTEMA 74
6.2.1 ESQUEMA DE MONTAJE 74
6.2.2 DIAGRAMA DE CASOS DE USO 75
6.2.3 DIAGRAMA DE COMPONENTES 76
6.2.4 DIAGRAMA DE CLASES 78
6.2.5 DIAGRAMAS DE SECUENCIA 79
6.2.6 DEFINICIÓN DEL SERVICIO 80
6.3 IMPLEMENTACIÓN DEL SISTEMA 83
6.3.1 SOFTWARE EN EL CLIENTE 83
6.3.2 SOFTWARE EN LA CÁMARA IP 84
PRESUPUESTO 85
CONCLUSIONES 87
GLOSARIO 89
BIBLIOGRAFÍA 91

ÍNDICE DE FIGURAS

Figura 1.1: Esquema de un sistema de vigilancia IP 2
Figura 2.1: Esquema de un conversor analógico digital 5
Figura 2.2: Diagrama de conversión RAW 6
Figura 2.3: Píxeles formando una imagen ByN y otra en Color 6
Figura 2.4: Software de gestión de cámaras IP 7
Figura 2.7: Cámara AXIS 210 seleccionada para pruebas 8
Figura 2.8: Arquitectura de una cámara IP 9
Figura 2.9: Diagrama de bloques del microcontrolador de la Cámara IP 10
Figura 2.10: Plataforma de aplicaciones de cámaras AXIS 14
Figura 2.11: AXIS Cross Line Detection 14
Figura 3.1 Imagen original para probar el contador de objetos 19
Figura 3.2 Vista de la imagen cargada en el espacio de trabajo de Matlab 19
Figura 3.3 Thresholding de una imagen ruidosa 20
Figura 3.4 Proceso de Thresholding 21
Figura 3.5 Segmentación de la imagen de prueba 23
Figura 3.6 Espacio de trabajo de Matlab tras el proceso de segmentación 23
Figura 3.7 Rellenado de huecos en la imagen de prueba 24
Figura 3.8 Erosión de un cuadrado usando una forma circular 24
Figura 3.9 Matriz con forma de diamante para el proceso de suavizado 25
Figura 3.10 Suavizado de bordes 25
Figura 3.11 Filtrado de objetos pequeños 26
Figura 3.12 Conectividad de pixeles 26
Figura 3.13 Etiquetado de una imagen 27
Figura 3.14 Imagen etiquetada resultado del proceso 28
Figura 3.15 Resultado del proceso contador de objetos 28
Figura 3.16 Proceso de etiquetado completo 29
Figura 3.17 Modelo Cliente 30
Figura 3.18 Modelo del Contador de Objetos 31
Figura 3.19 Inspecciones durante el proceso 31
Figura 3.20 Granos de arroz contados de forma manual 32
Figura 3.21 Generación de código desde Simulink 33
Figura 3.22 Máscara del filtro Quick Edge 38
Figura 3.23 Aplicación del filtro Quick Edge 39

Figura 3.24 Explicación de sobrescritura (0,0) 40
Figura 3.25 Detección de color por tolerancia 41
Figura 3.26 Detección de color por proporción 42
Figura 3.27 Comparativa de métodos de detección del color 43
Figura 3.28 Resultado en depuración del recuento BMP_FastCountColorProp 44
Figura 4.1 Arquitectura de Servicios Web 47
Figura 4.2 Torre de protocolos DPWS 48
Figura 4.3 Arquitectura DPWS 50
Figura 4.4 Dispositivo descubierto desde Windows Vista 53
Figura 4.5 Accediendo a la página web del dispositivo 53
Figura 4.6 Página web del dispositivo 54
Figura 4.7 Información del dispositivo desde DPWS Explorer 54
Figura 4.8 Estructura del WSDL 55
Figura 4.9 Información del servicio desde DPWS Explorer 56
Figura 4.10 Generación de código mediante gSOAP 57
Figura 4.11 Servidor a la espera de peticiones 58
Figura 4.12 Operación desde DPWS Explorer 58
Figura 4.13 Monitorización SOAP desde DPWS Explorer 59
Figura 4.14 Detalle del mensaje SOAP del evento 59
Figura 5.1 Mascara de Cambios en la detección de movimiento 62
Figura 5.2 Detección de movimiento con Matlab (Resultado) 63
Figura 5.3 Detección de movimiento con Matlab (Código 1/2) 63
Figura 5.4 Detección de movimiento con Matlab (Código 2/2) 64
Figura 5.5 Instantánea de ejemplo de un museo 65
Figura 5.6 Problemas durante la segmentación 65
Figura 5.7 Ejemplo de resultados del contador de personas 66
Figura 5.8 Detección de cara por segmentación de color 67
Figura 5.9 Detección mediante redes neuronales 68
Figura 5.10 Algoritmo básico para detección de caras mediante redes neuronales 68
Figura 5.11 Ejemplo de detección de patrones de Simulink 69
Figura 5.12 Correlación de un patrón contra la imagen de la cámara del portátil (1/2) 70
Figura 5.13 Correlación de un patrón contra la imagen de la cámara del portátil (2/2) 70
Figura 5.14 Algoritmo de segmentación por color mediante el sistema de inecuaciones . 71
Figura 5.15 Detección del color con sistema de inecuaciones 72
Figura 6.1 Esquema del control del nivel de llenado 73

Figura 6.2 Diagrama de secuencia en línea de producción 74
Figura 6.3 Diagrama de casos de uso 75
Figura 6.4 Diagrama de componentes 76
Figura 6.5 Diagrama de clases 78
Figura 6.6 Diagrama de secuencia del disparo 79
Figura 6.7 Diagrama de secuencia del servicio 79
Figura 6.8 Aplicación Setup 83
Figura 6.9 Restauración del backup con el software de la cámara 84

1 INTRODUCCIÓN

El objeto del proyecto se encuadra en los Sistemas de Vigilancia IP, no restringiendo su uso únicamente a la seguridad, sino también en la inspección visual de procesos industriales.

La inspección visual consiste en el procesado de imágenes digitales para extraer cierta información útil de la secuencia de imágenes. Pertenece a una de las ramas de la Inteligencia Artificial llamada Visión por Computador.

Actualmente la Visión por Computador en Sistemas de Vigilancia se realiza de forma centralizada en computadoras dedicadas. En instalaciones medianas ya se requiere gran potencia de cálculo y genera un consumo constante del ancho de banda de la red debido a la transmisión continua de un flujo de vídeo.

La idea de este proyecto consiste en incrustar una parte del proceso en la arquitectura de las Cámaras IP dotándolas de cierta inteligencia. Con esto se pretende reducir el ancho de banda consumido, transmitiendo sólo la información extraída al procesar las imágenes dentro de la cámara IP y distribuir así la carga del proceso.

1.1 PLANTEAMIENTO

Para ello se desarrollan componentes de Software Embebidos en las Cámaras IP de forma que puedan extraer cierta información de las imágenes y realizar acciones acorde con la información extraída o trasmitir dicha información mediante Arquitecturas SOA (Service Oriented Architecture) y así delegar la toma de decisiones en otros sistemas.

Para tomar consciencia del problema tomamos como ejemplo el sistema de vigilancia de un centro de exposiciones de 4 estancias, en el que hay una cámara por cada estancia. En la siguiente figura se muestra un esquema de este tipo de sistema.

Figura 1.1: Esquema de un sistema de vigilancia IP

Hay que realizar un proceso sobre las imágenes para contabilizar a los visitantes y realizar una estadística de la asistencia a eventos que incluya el interés mostrado en cada estancia. Este proceso no debe impactar en el sistema de vigilancia de seguridad.

Al ser una red IP no tenemos dificultad en añadir un cliente de visualización más a cada cámara. Un computador conectado en la red puede consumir la imagen de las 4 cámaras para realizar el proceso de forma independiente.

Una señal de video VGA 640x480 con una profundidad de color de 24 bits sin compresión a 30 cuadros por segundo ocupa un ancho de banda de 221,184 Mbit/s, sin tener en cuenta cabeceras de información necesarias para el formato de imagen ni las necesarias para su transmisión por la red. La velocidad habitual de una red LAN es de 100 Mbit/s, bajo estas condiciones no se puede transmitir ni siquiera una de las 4 señales de video.

Este problema se aborda generalmente introduciendo formatos de compresión de video, reduciendo la tasa de refresco, la resolución y/o la información de color.

Una señal de video VGA 640x480 a una velocidad de refresco de 30 cuadros por segundo codificada con MPEG-4 con una compresión del 30% ocupa un ancho de banda de 603 Kbit/s. Para el ejemplo de las 4 cámaras, el computador cliente realiza un consumo de 2,41 Mbit/s.

Esta es una solución válida cuando el que va a extraer la información es un humano, pero cuando la imagen debe ser procesada por un computador surgen diversos problemas como el aumento de la potencia de cálculo requerida para interpretar la señal comprimida, el aumento considerable del ruido de imagen y perdida de la información de color. Estas aberraciones de la

imagen obligan a introducir contramedidas en el algoritmo de visión por computador aumentando nuevamente la potencia de cálculo requerida.

Es más razonable transmitir únicamente la información útil "11:45, 28 personas, sala 3" en vez de transmitir toda la imagen para procesarla. Para ello hay que considerar las limitaciones de hardware de las cámaras existentes en el mercado. Actualmente, las prestaciones de la mayoría de cámaras IP del mercado permiten llevar a cabo un proceso de imágenes incorporando hardware dedicado (DSP - Digital Signal Processor - y compresores JPG y MPEG) a una velocidad más que aceptable.

1.2 OBJETIVOS CONSEGUIDOS

Se ha conseguido realizar de forma práctica la incrustación del componente de procesado de imagen en la cámara. Muchos de los obstáculos encontrados han resultado ser específicos de la plataforma software de la cámara seleccionada para las pruebas y prácticamente todos han sido atajados con una solución alternativa.

La gran mayoría de librerías de procesado de imágenes y visión por computador están escritas en lenguaje C++ pero la librería estándar de C presente en la cámara, la librería *uClibc* no permite su compilación y presenta una incompatibilidad con los caracteres de 16 bit (tipos *wchar_t*) impidiendo la distribución de la información con arquitecturas SOA, siendo una parte fundamental de la rama Telemática. Este aspecto se aborda teóricamente para facilitar futuros trabajos sobre este campo de investigación y será reemplazado en la práctica por una solución básica.

La cámara a nivel hardware tiene las capacidades suficientes para realizar el caso de uso planteado cumpliendo todas las expectativas del proyecto y por lo tanto **la idea del proyecto es factible**.

2 DESCRIPCIÓN DEL MARCO DE TRABAJO

2.1 CÁMARAS IP

Las cámaras IP son cámaras digitales que utilizan la red IP para transportar sus señales de imagen y sonido, además de poder recibir comandos de control.

Entre muchas de sus ventajas permiten:

- Visualizar la imagen desde cualquier sitio del mundo gracias a Internet.
- Reducir los gastos de instalación ya que se puede utilizar la red de datos del edificio, ya sea cableada o inalámbrica.
- Ganar en calidad al ser una señal digital.

2.1.1 CÁMARAS DIGITALES

Las cámaras digitales utilizan un sensor de imagen para transformar la luz recibida en una señal digital de unos y ceros. El sensor está compuesto de una parrilla de pequeños sensores fotoeléctricos (celdas) que generan una señal eléctrica en función de la luz recibida (al igual que los conos y bastones de la retina de los mamíferos). Esta señal eléctrica es muestreada y convertida a digital mediante un convertidor analógico digital (ADC) como el de la Figura 2.1: Esquema de un conversor analógico digital.

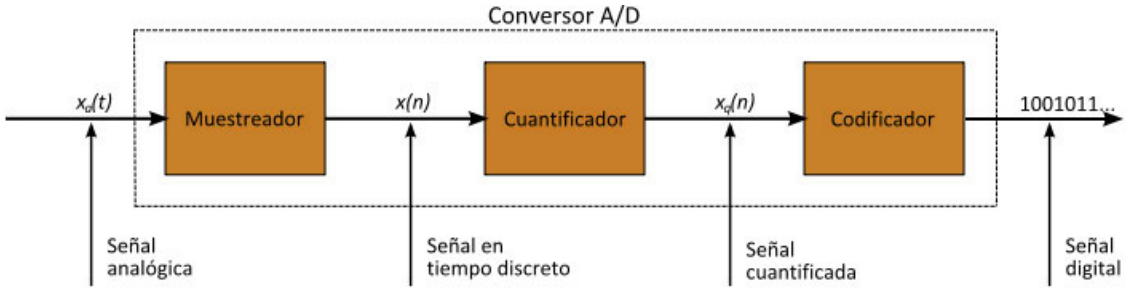


Figura 2.1: Esquema de un conversor analógico digital

Existen varios tipos de sensores pero para el caso que nos compete, los de color, utilizan una parrilla cuadrada con una máscara superpuesta de filtros de los colores primarios rojo, verde y azul. Al igual que en la retina hay mayor número de células sensibles al verde (bastones), generalmente se usa una máscara (25% rojo, 25% azul y 50% verde) denominada Filtro Bayer en honor a su inventor Bryce E. Bayer. Esta máscara permite sacrificar resolución a favor de obtener color sin perder luminosidad. La imagen tal como la obtiene el sensor se denomina RAW (crudo), posteriormente un procesador de imagen interpola la información de cada canal de color para estimar el valor de los pixeles tapados con el filtro de otro color. Adicionalmente estos procesadores realizan otras funciones como el balance de blancos, filtro de ruido, compresión, etc.; para finalmente obtener una imagen en un formato especifico, generalmente JPEG o TIFF. Se puede ver un diagrama de este proceso en la Figura 2.2: Diagrama de conversión RAW.

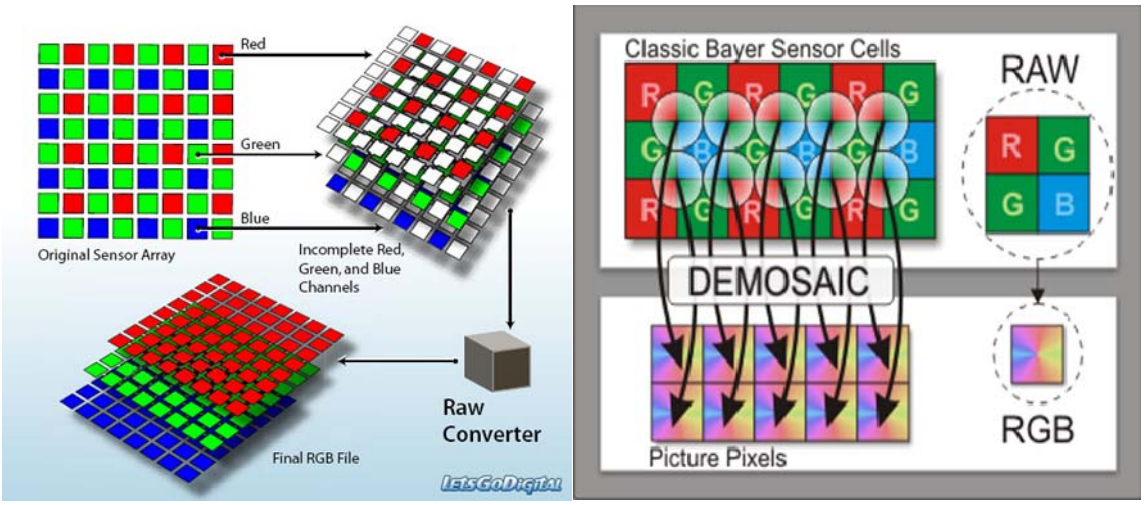


Figura 2.2: Diagrama de conversión RAW

La imagen final se caracteriza por tener una resolución igual al número de pixeles horizontales por el número de pixeles verticales. Se mide comercialmente en MegaPíxeles (10^6 Píxeles). Un pixel es un elemento de color uniforme, con gran cantidad de pixeles se puede formar una imagen a modo de mosaico como muestra la figura 2.3. Los pixeles en color tienen tonalidades para el rojo, verde y azul. Dependiendo del proceso de cuantificación en el ADC se obtienen diferentes niveles de color. El caso más extendido es usar 8 bits por pixel (bpp) para cada canal de color. Para las imágenes en color se usa una profundidad de color de 24bpp (8 rojo + 8 verde + 8 azul) aunque esto depende del formato de imagen.

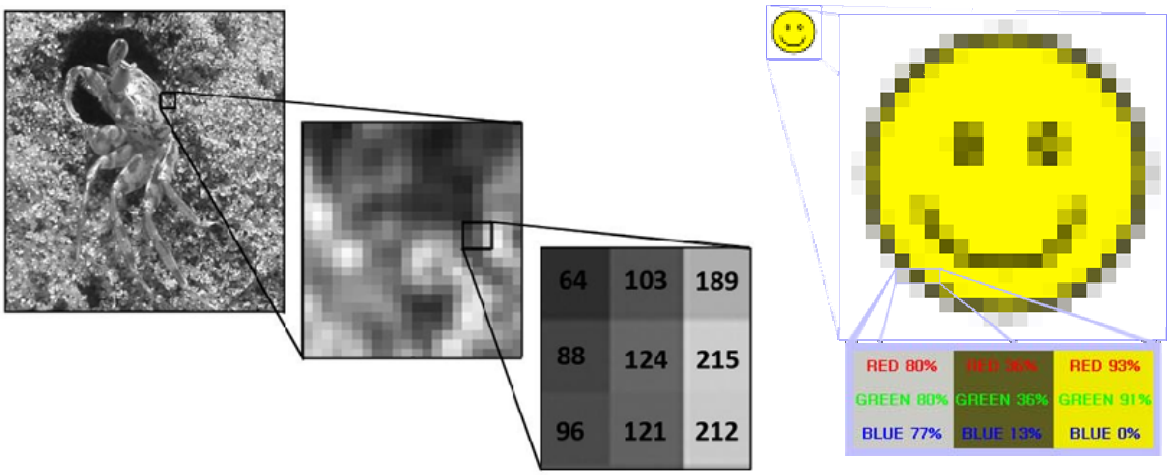


Figura 2.3: Píxeles formando una imagen ByN y otra en Color

2.1.2 SISTEMAS DE VIGILANCIA IP

Las redes Ethernet son indispensables en cualquier edificio donde se quiera tener acceso a Internet. Cualquier edificio con una red Ethernet donde se quiera instalar un sistema de vigilancia lo tiene muy fácil gracias los Sistemas de Vigilancia IP. En cualquier lugar donde haya

una toma de Ethernet se puede instalar una cámara de vigilancia y la imagen se puede visualizar y grabar desde cualquier punto de la red o desde Internet. Además hay cámaras capaces de conectarse de forma inalámbrica a un punto de acceso WiFi de la red del edificio.

Los gastos materiales del sistema se reducen a los gastos de adquisición de las cámaras IP. Este tipo de sistemas no captan ruido durante la transmisión por tratarse de sistemas digitales y no necesitan amplificadores ni repetidores como puede necesitar un sistema CCTV clásico.

Cualquier ordenador con un navegador web o el software de gestión proporcionado por el fabricante (ver Figura 2.4: Software de gestión de cámaras IP) puede acceder a la imagen de las cámaras. Por supuesto se puede establecer unos niveles de acceso mediante contraseña y la transmisión está cifrada con HTTPS.

Figura 2.4: Software de gestión de cámaras IP

Aparte de la simple transmisión y grabación del video algunas cámaras IP ofrecen la posibilidad de lanzar alarmas en función de la actividad visualizada, mandar correos electrónicos, tomar fotografías y subirlas a un directorio FTP, ocultar una zona de la imagen, etc.

Una función muy útil y ventajosa es la capacidad de iniciar la grabación únicamente cuando se detecta movimiento en una o varias regiones de la imagen.

El objetivo de este proyecto es desarrollar una nueva función sobre la actividad visualizada utilizando algoritmos de procesamiento de imágenes.

2.1.3 RED IP

Gracias a la red IP una compañía puede controlar la seguridad de varios emplazamientos en distintas partes del mundo desde una sede central sin demasiadas complicaciones. Al estar la red Ethernet de los edificios conectada a Internet mediante un router se puede realizar la configuración NAT para tener acceso a las diferentes cámaras del edificio desde cualquier parte

del mundo. Las limitaciones vendrán en función de la velocidad de conexión a Internet que tengan los edificios.

Existen en el mercado servicios de grabación externos que se benefician de la conectividad IP para guardar las grabaciones de forma remota. Este tipo de servicios añade seguridad adicional al mantener alejado el material de grabación del edificio cuya seguridad se vea comprometida, por ejemplo en un incendio el sistema de grabación puede resultar dañado.

2.1.4 SELECCIÓN DE UNA CÁMARA PARA PRUEBAS

Durante el desarrollo del presente Proyecto de Fin de Carrera se ha trabajado con cámaras IP del fabricante AXIS Communications por disponer de un firmware Linux embebido y proporcionar herramientas para el desarrollo tanto de programas de control, externos a la cámara, como de programas embebidos en la propia cámara.

Concretamente se ha trabajado con la Cámara IP AXIS 210 por ser de las económicas la que mayores capacidades hardware ofrece.

Figura 2.5: Cámara AXIS 210 seleccionada para pruebas

Características del producto

Las cámaras AXIS 210/211 están basadas en el chip de compresión AXIS ARTPEC-2.

Ambos modelos soportan las siguientes características:

- Soporte simultáneo de secuencias Motion JPEG y MPEG-4 para la optimización de imagen y ancho de banda.
- Detección de movimiento integrada, con buffer de pre y post alarma. Que se puede utilizar para iniciar la grabación cuando se produce actividad en la imagen de video.
- Grabaciones programadas en determinados momentos.
- Filtrado de direcciones IP y protección con contraseñas multinivel.
- Entrada y salida de señales digitales de alarma, que pueden conectarse con diversos dispositivos externos como sensores de apertura de puertas y campanas de alarma.

- Múltiples resoluciones de video.
- Acceso simultaneo de hasta 20 espectadores cuando se usa Motion JPEG. El número de espectadores es ilimitado cuando se utiliza multicast MPEG-4, pero cada espectador necesita una licencia MPEG-4. Se incluye una licencia con el producto, pero se pueden adquirir licencias adicionales.
- Se pueden configurar hasta 3 aéreas de privacidad en las cuales no se muestra la imagen.
- El servidor integrado permite el acceso completo a través de un navegador web estándar.
- La herramienta de creación de scripts integrada le permite la creación de aplicaciones básicas. Para una funcionalidad avanzada, se puede acceder a la cámara a través del API AXIS HTTP.

Las características adicionales para la AXIS 211 son las siguientes:

- La AXIS 211 tiene un Iris-DC de distancia focal variable, que regula automáticamente la cantidad de luz que entra en la cámara. El enfoque y la profundidad de campo se ajustan manualmente con la ayuda de unas manecillas montadas en el objetivo.
- Se puede alimentar usando PoE conforme estándar IEEE 802.3af.

2.1.5 ARQUITECTURA DE LA CÁMARA IP

Una cámara IP se puede describir como una cámara y un ordenador combinados. Tienen un chip de compresión, un sistema operativo, un servidor web interno, un servidor FTP, un cliente FTP, cliente e-mail, control de alarmas y mucho más.

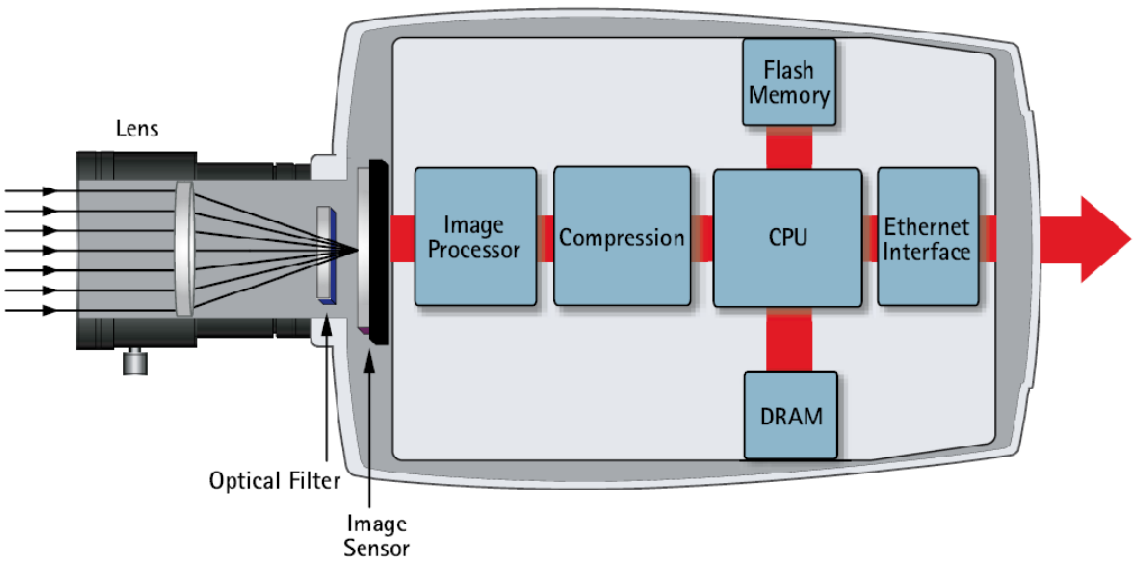


Figura 2.6: Arquitectura de una cámara IP

Como se observa en el diagrama la escena se proyecta sobre el sensor de imagen gracias a un juego de lentes, la señal del sensor, en crudo, es interpretada por el procesador de Imagen y

luego se le aplican los correspondientes algoritmos de compresión, la imagen procesada se transmite usando la interfaz Ethernet. Todo ello está gobernado por el procesador de la cámara.

La cámara seleccionada tiene un microcontrolador ETRAX 100LX de 32bit basado en el ETRAX 100 con el añadido de una unidad de manejo de memoria (MMU) que le permite funcionar con Linux, de ahí el prefijo LX. A continuación se muestra un diagrama de bloques de la arquitectura del microcontrolador.

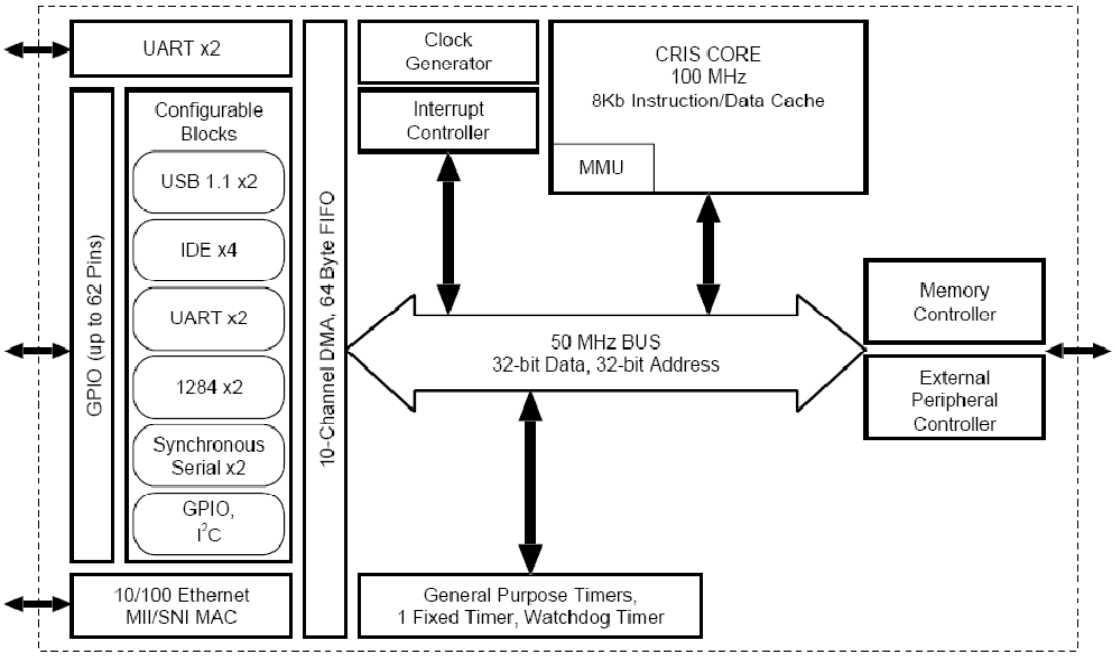


Figura 2.7: Diagrama de bloques del microcontrolador de la Cámara IP

El microcontrolador lleva un procesador RISC de 100 MIPS, una cache unificada de instrucciones y datos de 8Kb, Puertos de entrada/salida controlados por DMA y un controlador de Fast Ethernet. El procesador del ETRAX 100LX es un procesador RISC de 32-bit con instrucciones de 16-bit que cumple con el conjunto reducido de instrucciones (CRIS) de Axis. Funciona a una frecuencia de 100 MHz, dando un rendimiento pico de 100 MIPS.

La cámara Axis 210 viene equipada con 4 MB de memoria Flash, 16 MB de memoria RAM de los cuales 1,2 MB se utilizan para los buffer de las alarmas.

2.1.6 FIRMWARE

El software que ejecuta la cámara (firmware), puede ser actualizado de una manera sencilla desde la propia interfaz web de la cámara. El fabricante publica actualizaciones periódicas del firmware para añadir nuevas funcionalidades y corregir errores encontrados. En el caso de las cámaras AXIS el fabricante proporciona las utilidades y el kit de desarrollo de software (SDK) para que cualquiera pueda realizar ampliaciones de software sobre la cámara.

El fabricante AXIS ha seleccionado el sistema operativo Linux como base para sus desarrollos sobre la cámara. Ha seleccionado Linux por la existencia de numerosas librerías de código abierto para integrar en sus productos. A continuación se analizan las piezas de software disponibles en el Firmware del fabricante de las cámaras.

2.1.7 LINUX EMBEBIDO

Las cámaras AXIS llevan una distribución Linux de versión igual o superior a la 2.4 con una librería estándar de C de reducido tamaño denominada uClibc (micro C lib) y con la Shell dash. La AXIS 210 concretamente lleva la versión 2.6.

Axis pone a disposición del público un subconjunto del entorno de desarrollo utilizado por el fabricante, este entorno de desarrollo no incluye el software específico de los productos, pero sirve como base para realizar el desarrollo de aplicaciones empotradas. Existen varias versiones del entorno de desarrollo de software (SDK), para realizar el presente proyecto se ha utilizado la versión 2.20 con algunas modificaciones de la configuración que se comentarán más adelante.

El listado completo de las aplicaciones disponibles en la distribución se puede obtener en la referencia [4]. Por mencionar algunas: tiene el servidor web boa con soporte para CGI, el cojunto de utilidades busybox, la reducida shell POSIX dash, el asistente para depuración remota gdbserver, encriptación de sockets con openSSL, el cliente FTP sftpclient, el cliente SMTP smtpclient, el editor de ficheros de texto vía web editcgi, el servidor telnet utelnetd, el servidor FTP vftpd, etcétera.

2.1.7.1 DESARROLLO DE APLICACIONES EMPOTRADAS

Primero hay que mencionar que no se puede desarrollar sobre Windows usando cygwin porque varias piezas del SDK asumen que se está usando GNU/Linux. Por eso se ha utilizado una maquina virtual con Ubuntu pre-instalado donde se ha instalado el compilador facilitado por AXIS y el entorno de desarrollo SDK v2.20.

La arquitectura de la cámara (CRIS) es distinta que la del entorno de desarrollo utilizado (x86) por lo tanto hay que realizar compilaciones cruzadas del código, lo que dificulta en cierta medida las pruebas. La arquitectura CRIS es soportada por el compilador gcc. La versión del compilador gcc distribuido por Axis (v1.64) es el único sobre el que se puede obtener soporte oficial por parte de AXIS Communications. Se instala por defecto en la ruta `/usr/local/cris`.

Realmente consiste en dos compiladores uno para crisv10 (diseños basados en ETRAX 100LX) y uno para crisv32 (diseños basados en ETRAX FS). El script `gcc-cris` ayuda a llamar al compilador apropiado teniendo en consideración la librería estándar de C seleccionada. Para la cámara AXIS 210 es necesario compilar usando el compilador `cris-axis-linux-gnuuclibc`. Esto

significa que la cámara tiene la librería uclibc y eso presenta una serie de inconvenientes que se relatarán en la sección de problemas encontrados.

Para realizar el "Hola Mundo" resulta útil consultar la referencia [5]. El siguiente paso lógico es preparar un makefile, para ello conviene consultar la referencia [6].

Alterar el firmware puede traer consecuencias catastróficas, sobre todo porque en el sdk no se incluye todo el código que va en la cámara. Por otro lado la memoria flash de la cámara tiene una vida limitada a 100.000 escrituras. Para pruebas se recomienda transferir los ficheros por Telnet y usar la partición montada en la memoria RAM (/tmp) teniendo en cuenta que al apagar la cámara se borra el contenido de la RAM.

Las pruebas de compilación cruzada exigen montar un entorno de simulación en código, dado que la máquina virtual de desarrollo no tiene cámara para obtener una imagen, ni un led bicolor para encenderlo, apagarlo o cambiarlo de color. Para ello se han utilizado macros del compilador para distinguir el destino de la compilación, `host` para la máquina virtual y `cris-axis-linux-gnuuclibc` para la cámara. También se han montado dos scripts de compilación uno para cada destino que invocan al makefile con distintos parámetros `compilar` y `compilar_host`.

```
#! /bin/sh

echo "Cargando variables de entorno"
. ../../devboard-R2_20/init_env
. ./init_env
echo "Definiendo el target cris-axis-linux-gnuuclibc"
make cris-axis-linux-gnuuclibc
echo "Limpiando compilacion anterior (Rebuild)"
make clean
echo "Ejecutando make"
make
echo "Instalando librerias en /lib"
cp *.so ../lib
echo "Proceso terminado"
```

Código 2.1: Script de compilación para la cámara

En el caso de la imagen, se han obtenido unas cuantas imágenes para pruebas de la propia cámara. Se han tomado las instantáneas y se han descargado vía ftp para tenerlas en el mismo formato. En el proceso de compilación se sustituye el comando de tomar la instantánea por un mensaje sacado por consola. A continuación se expone la macro utilizada para simulación.

```
#ifndef HOST
        //take a snapshot
        system("/bin/bitmap 2 bmp > snapshot.bmp");
#else
        //always the same snapshot
        printf("Simulating to take a snapshot\n");
#endif
```

Código 2.2: Simulación de la instantánea

El código ejecutado en la cámara hay que probarlo de forma independiente. La definición de la etiqueta HOST se realiza en el makefile y se pasa como parámetro al compilador `CFLAGS += -`

`DHOST`. El SDK define el target en el fichero `.target-makefrag` para poder consultarlo en los diversos makefiles del árbol de directorios para decidir si se añade o no la etiqueta HOST. A continuación se puede observar uno de los makefiles utilizados.

```
AXIS_USABLE_LIBS = GLIBC UCLIBC
include $(AXIS_TOP_DIR)/tools/build/rules/common.mak

CFLAGS     += -fPIC -I../QDBMP
LDFLAGS    += -fPIC -shared -L../QDBMP
LDLIBS     = -lqdbmp
NOMBRE_LIB  = libIQCC.so.1.0.0
SONAME      = libIQCC.so.1.0

ifeq ($(strip $(AXIS_TARGET_OS)),host)
CFLAGS += -DHOST
endif

all: $(NOMBRE_LIB)

$(NOMBRE_LIB): QualityComponent.o
	$(CC) -Wl,-soname,$(SONAME) $(LDFLAGS) $(LDLIBS) -o $@ $^
	ln -s $(NOMBRE_LIB) libIQCC.so
	ldconfig -vn ./

#$(PROGS): $(OBJS)
#	$(CC) $(LDFLAGS) $^ $(LDLIBS) -o $@

#install: $(PROGS)
#	$(INSTALL) -d $(INSTDIR)
#	$(INSTALL) -m $(INSTMODE) -o $(INSTOWNER) -g $(INSTGROUP) $(PROGS) $(INSTDIR)

clean:
	rm -f *.o core libIQCC.so*
```

Código 2.3: Makefile del componente de calidad

Este makefile está preparado para generar la librería dinámica `libIQCC.so` que a su vez utiliza la librería dinámica `qdbmp.so`. Al usar librerías dinámicas se pueden actualizar fragmentos del código sin tener que recompilar todo. Para realizar este makefile se ha consultado la referencia [8].

2.1.8 PLATAFORMA DE APLICACIONES DE CÁMARAS AXIS

Existen nuevas cámaras AXIS que disponen de un sistema sencillo de extensión de la funcionalidad. La Plataforma de aplicaciones para cámaras AXIS permite el desarrollo de aplicaciones de terceros que se pueden descargar e instalar en cámaras de red y codificadores de vídeo de Axis. Este sistema apareció mientras se realizaba el proyecto, en caso contrario, se hubiera intentado abordar mediante esta solución.

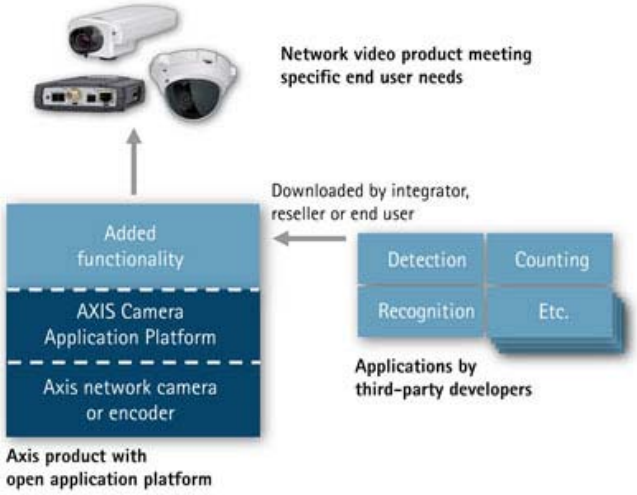


Figura 2.8: Plataforma de aplicaciones de cámaras AXIS

La plataforma gestiona un sistema de instalación y un sistema de licencias. Por ejemplo, una aplicación disponible es la AXIS Cross Line Detection. Es sencilla pero versátil, permite establecer una alarma cuando un objeto cruza una o varias líneas definidas sobre la propia imagen. Aquí se muestran unos cuantos casos de uso de la aplicación.

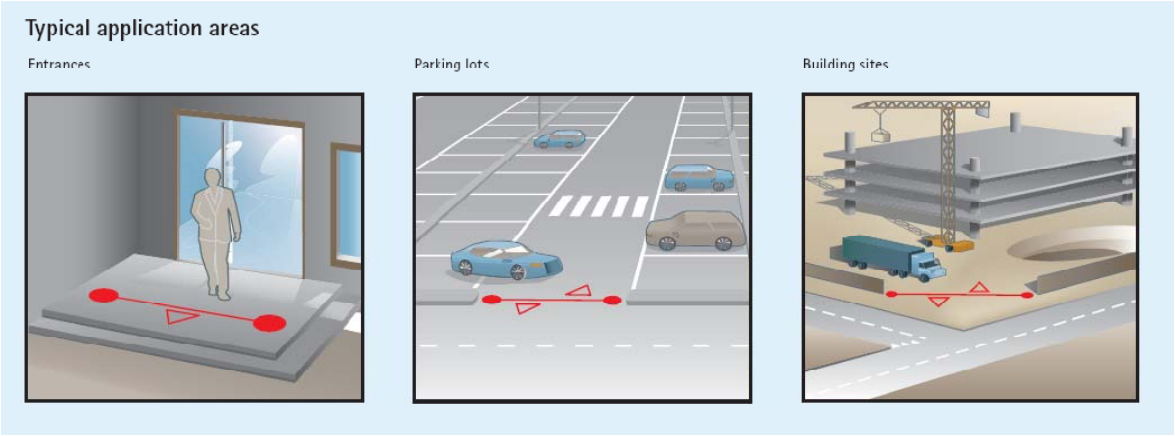


Figura 2.9: AXIS Cross Line Detection

2.2 PROBLEMAS ENCONTRADOS

La instalación del entorno de desarrollo no ha sido trivial. Han surgido por incompatibilidades entre las versiones de las librerías instaladas en la cámara y las disponibles en el entorno de desarrollo. La última versión del SDK, la v2.20 no permite compilar porque tiene una versión superior de la uclibc. Al intentar instalar la versión anterior del SDK, la v2.10, presenta problemas de instalación.

Esta dificultad se soluciona editando los ficheros de configuración de la instalación del SDK 2.20 para que utilicen la librería uclibc en la versión requerida 0.9.27-15. La librería de FTP también hay que bajarla a la versión 1.0.1. Durante la instalación se descargan e instalan las librerías especificadas en los ficheros de configuración `/configure-files/common/common-tag*`.

Uno de los grandes problemas del proyecto derivados del uso de la librería uclibc es la inviabilidad de compilar código escrito en C++ llevando a descartar numerosas librerías de procesado de imagen por estar escritas en C++. Aunque seguramente se hubieran tenido que descartar por las limitaciones de espacio en la cámara.

Aparte del SDK hay que instalar algunas librerías auxiliares que no están claramente especificadas, se van descubriendo a base de errores de instalación.

3 PROCESAMIENTO DE IMÁGENES

En este apartado se explica cómo se diseña y materializa un algoritmo de procesamiento de imágenes. Se explican una serie de pasos comunes en los algoritmos de procesado y cómo se ajustan durante el diseño. También se explica cómo generar código a partir de los diseños y cómo usar librerías ya existentes.

3.1 INTRODUCCIÓN

Cuando se habla de procesado de imágenes se hace referencia a imágenes digitales. Independientemente del formato de los archivos se pueden dividir en 2 casos generales, imágenes estáticas y secuencias de video. En el caso de las imágenes estáticas la información está almacenada en una matriz bidimensional de pixeles, como se mostró en el capítulo anterior, y el caso del video no es nada más que una secuencia de imágenes estáticas llamadas frames o cuadros, formando una matriz tridimensional de pixeles. La diferencia importante es que ciertos algoritmos de procesado utilizan información de distintos cuadros y eso supone una ventaja con respecto a las imágenes estáticas. Por ejemplo, los algoritmos de compresión de video reducen el tamaño del archivo gracias a que pueden hacer referencia a variaciones con respecto a frames anteriores.

El procesado de imágenes no es otra cosa que la transformación de esas matrices de pixeles en información útil de forma automática. Puede transformarse una imagen en otra, una secuencia de imágenes en una sola imagen y la salida no tiene por qué ser una nueva imagen. Siendo generalistas, se extraen nuevos datos.

El algoritmo de procesado se encarga de hacer dicha transformación. Consiste en una secuencia de operaciones matemáticas que se llevan a cabo sobre la matriz de entrada.

3.2 DESARROLLO DE UN ALGORITMO

Dado que un algoritmo de procesado de imágenes es una secuencia de operaciones matemáticas lo más lógico para diseñar un algoritmo es usar software especializado en matemáticas. Este tipo de programas están optimizados para el trabajo con matrices. De hecho todo son matrices, un solo número es una matriz 1x1, una secuencia de sonido es una matriz 1xN, una imagen es una matriz MxN.

Existen numerosos programas de cálculo matemático, algunos del mundo libre como GNU Octave o Scilab, algunos son de pago como Mathematica o Matlab. Para conocer una comparativa consultar la referencia [9].

Se ha empleado Matlab por ser una herramienta muy extendida y haber sido utilizada con anterioridad en otras asignaturas de la titulación.

3.2.1 MATLAB

La ayuda incluida en Matlab es una completa base de datos del conocimiento. Se han utilizado varios fragmentos del código disponible en la ayuda para la realización de varios estudios. A continuación se muestran varios procesos elementales que al combinarlos permiten realizar un proceso de mayor complejidad. Se explica un algoritmo completo y se profundiza en sus procesos elementales.

3.2.1.1 Contador de objetos

Se ha buscado un algoritmo que pueda ofrecer una información numérica para ser consumida como un servicio. Contar los objetos presentes en una imagen, bajo determinadas condiciones, es un proceso sencillo.

Es interesante contar el número de personas presentes en una estancia de un museo por ejemplo para conocer las áreas de mayor interés, recolocación del material expuesto, registro del nivel de aforo, etc. En procesos industriales se puede usar para realizar un control de calidad y verificar automáticamente que un paquete contiene el número indicado de piezas.

Los pasos a realizar en este algoritmo son los siguientes:

Paso 1: Lectura del fichero

En este paso se carga en memoria la imagen a procesar.

Paso 2: Conversión a mapa de bits

Separar los objetos del fondo. Aprovechando el contraste de la imagen.

Paso 3: Rellenar huecos

Recupera pixeles interiores que forman parte del objeto y no se han reconocido.

Paso 4: Suavizado de bordes

Redondear aristas en la imagen para suavizar el contorno de los objetos.

Paso 5: Filtrado por tamaño

Eliminar virutas pequeñas.

Paso 6: Etiquetado y conteo

Contabilizar bloques de pixeles.

Una vez repasados los pasos que hay que realizar, se detallan a continuación:

Paso 1: Lectura del fichero

La lectura del fichero en matlab se realiza con el siguiente comando que recibe la ruta en disco del fichero y crea la matriz correspondiente en el espacio de trabajo.

```
I = imread(fichero);
```

La imagen leída se almacena en un array tridimensional, en cada dimensión se codifica el nivel de intensidad de cada pixel correspondiente a los tres canales de color RGB.

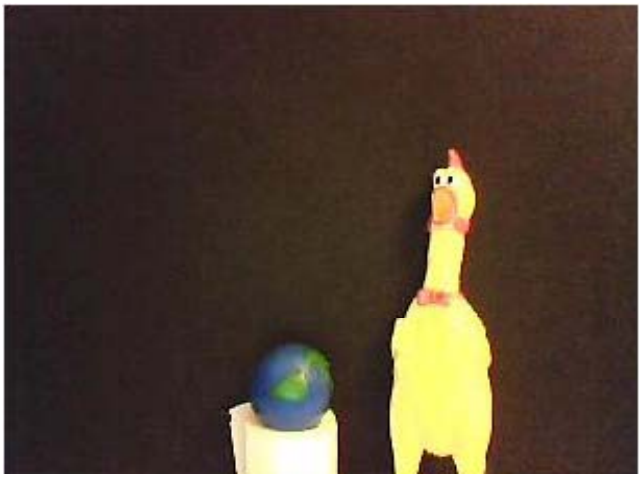


Figura 3.1 Imagen original para probar el contador de objetos

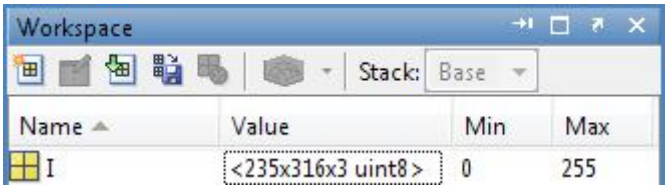


Figura 3.2 Vista de la imagen cargada en el espacio de trabajo de Matlab

En el espacio de **trabajo**, la imagen tiene unas dimensiones 235x316 y 3 canales de color. La intensidad en cada canal está codificada con enteros sin signo de 8 bits (uint8) que van del valor 0 al 255. 0 para el mínimo de intensidad y 255 para el máximo.

Paso 2: Conversión a mapa de bits

Para ello se utiliza un proceso elemental llamado segmentación que consiste en separar/segmentar los diferentes objetos presentes en una imagen utilizando diferentes características de la imagen. El proceso más sencillo de segmentación utiliza el contraste del objeto con respecto al fondo, si existe cierto grado de contraste es fácil aislar el objeto con un simple proceso de **Thresholding**.

Thresholding / valor umbral:

Proceso que trabaja sobre una imagen en escala de grises asignando el valor 1 a los pixeles de la imagen si superan cierto umbral de intensidad y 0 en caso contrario, resultando en una imagen binaria.

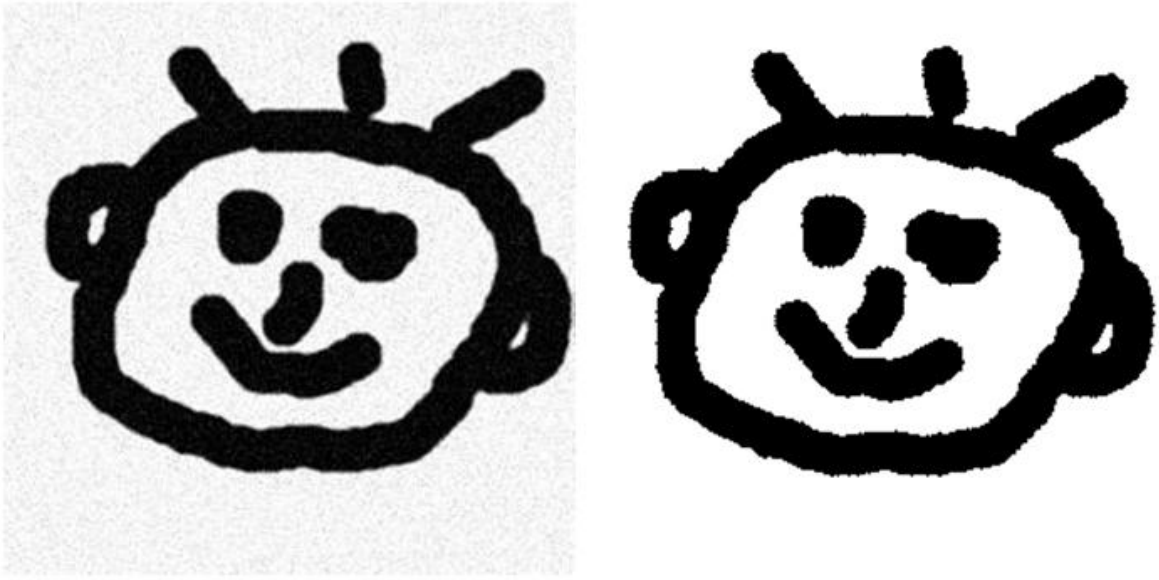


Figura 3.3 Thresholding de una imagen ruidosa

Este proceso elemental tiene varios usos, en la imagen de ejemplo se utiliza para eliminar el ruido en una imagen de alto contraste. De igual manera es utilizado en procesos de reconocimiento de texto escaneado o fotografiado, lectura mediante imagen de códigos de barras o códigos bidimensionales.

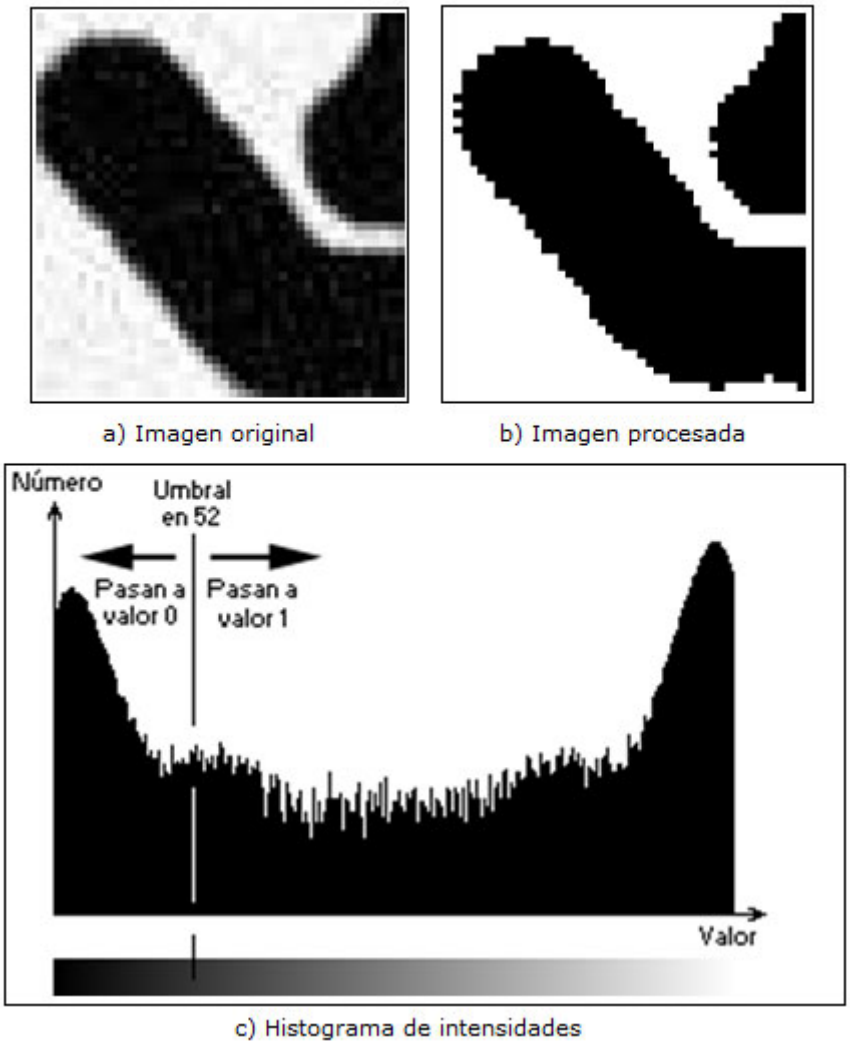


Figura 3.4 Proceso de Thresholding

Se puede observar en el histograma de la imagen que la mayoría de intensidades se sitúan en el negro y en el blanco, debido a que es una imagen con un alto contraste. Cortando en el umbral 52 se separa el objeto del fondo blanco como se puede ver en la imagen procesada.

En Matlab este proceso se realiza mediante un operador de comparación que trabaja sobre cada pixel de la matriz ImagenOriginal dejando el resultado de la operación en otra matriz ImagenProcesada:

```
ImagenProcesada = ImagenOriginal > Umbral; /%intensidades mayores que umbral pasan a 1
```

Si requiere ser programarlo en C hay que recorrer la imagen original al completo, pixel a pixel aplicando el operador lógico de la siguiente manera:

```
for (int i=0; i<ImagenOriginal.Width; i++)
   for (int j=0; j<ImagenOriginal.Height; j++)
      ImagenDestino[i][j] = (ImagenOriginal[i][j] > Umbral);
```

Conversión a escala de grises:

Muchos procesos, como el anterior, trabajan sobre imágenes en **escala de grises**. Existe el proceso de conversión RGB a BW que trabaja sobre los 3 canales de color calculando su apariencia en tonalidades de gris.

El proceso se basa en la visión de los humanos, los animales tienen diferentes sensibilidades a cada color, de hecho existen numerosas enfermedades de la visión cromática. En general se utiliza una formula normalizada según ITU-R [13] para realizar la conversión. Consiste en una media ponderada que asigna un peso a cada componente de color. El peso asignado está relacionado con la sensibilidad de las células del ojo a dichas longitudes de onda.

$$D_Y 2 = \mathrm{INT}\left[\ 0.299\ D_R 2 + 0.587\ D_G 2 + 0.114\ D_B 2\ \right]$$

```
Y = 0.299*R + 0.587*G + 0.114*B
```

En Matlab se realiza dicha conversión con el comando:

```
I = rgb2gray(RGB)
```

A continuación se ven las implementaciones del comando con bucle y sin bucle:

```
GIm=uint8(zeros(size(Im,1),size(Im,2)));
for i=1:size(Im,1)
      for j=1:size(Im,2)
          GIm(i,j)=0.299*Im(i,j,1)+0.587*Im(i,j,2)+0.114*Im(i,j,3);
      end
end
```

```
GIm=0.299*Im(:,:,1)+0.587*Im(:,:,2)+0.114*Im(:,:,3); %Conversión sin bucle
```

Como los procesos de conversión a escala de grises y el valor umbral son muy utilizados, hay un comando que los realiza de forma consecutiva y es el que se ha utilizado en este paso:

```
BW = im2bw(RGB, level)
```

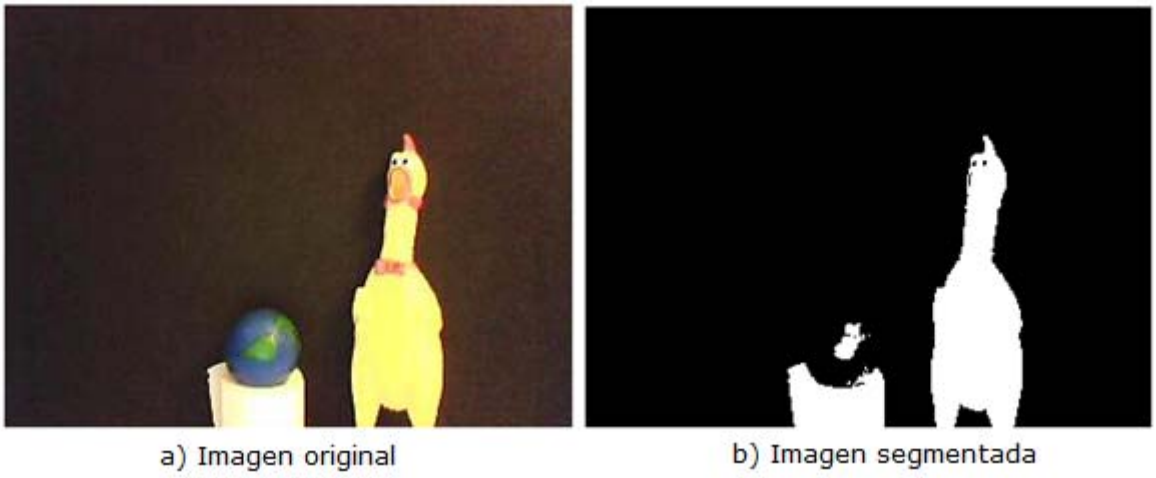


Figura 3.5 Segmentación de la imagen de prueba

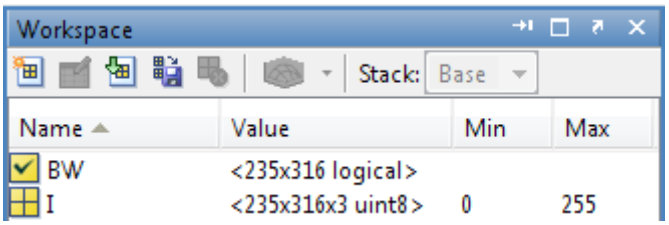


Figura 3.6 Espacio de trabajo de Matlab tras el proceso de segmentación

En el espacio de trabajo la imagen BW tiene las mismas dimensiones que la imagen original pero ha perdido los canales de colores, o mejor dicho ahora solo tiene uno y también ha perdido la gama de intensidades quedando una imagen binaria (logical) 1 y 0. Donde 0 es el mínimo de intensidad y 1 el máximo. A este tipo de imagen se le llama mapa de bits porque cada pixel está codificado con un único bit de información.

Paso 3: Rellenar huecos

El proceso de rellenado de huecos no es demasiado crítico para el contador de objetos, pero al medir el volumen de un objeto se puede perder volumen por los huecos.

En Maltab se realiza utilizando este comando:

```
BW = imfill(BW, 'holes');
```

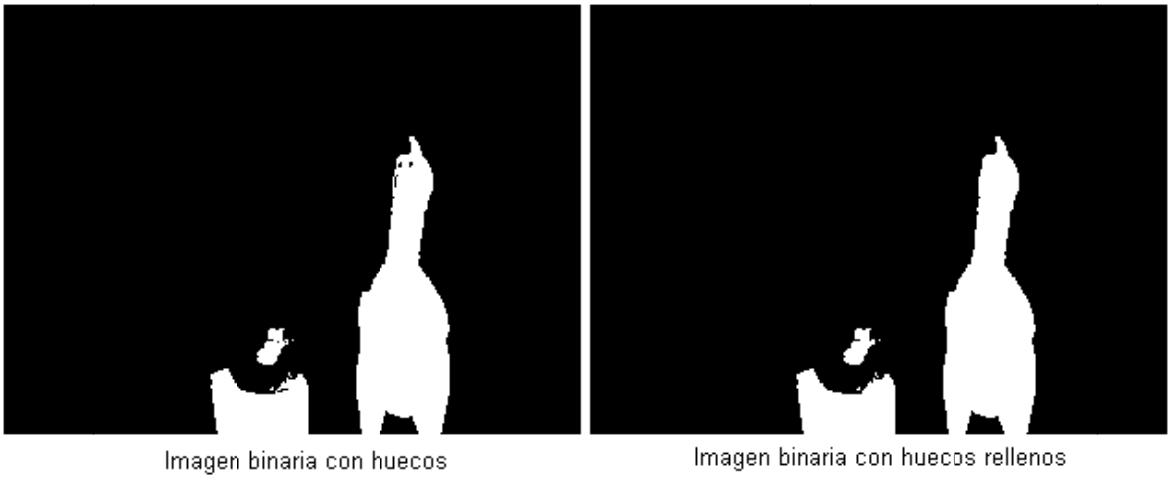


Figura 3.7 Rellenado de huecos en la imagen de prueba

Tras este paso los ojos del pollo forman parte del objeto.

Paso 4: Suavizado de bordes

Este paso es crítico para el contador de objetos, puesto que elimina las virutas de la imagen que, de no hacerlo, se cuentan como objetos pequeños falseando la medida. El proceso se conoce como **Erosión**.

Erode / erosión:

Proceso que trabaja sobre una imagen binaria aplicando una operación morfológica, cambia pixeles blancos por negros si se cumple determinada condición en una ventana de pixeles.

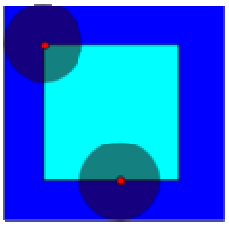


Figura 3.8 Erosión de un cuadrado usando una forma circular

Se realiza con el siguiente comando:

```
seD = strel('diamond',1); %forma utilizada
BW = imerode(BW,seD);
```

Utiliza una ventana en forma de diamante de radio 1:

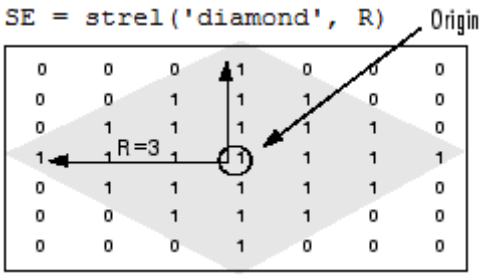


Figura 3.9 Matriz con forma de diamante para el proceso de suavizado

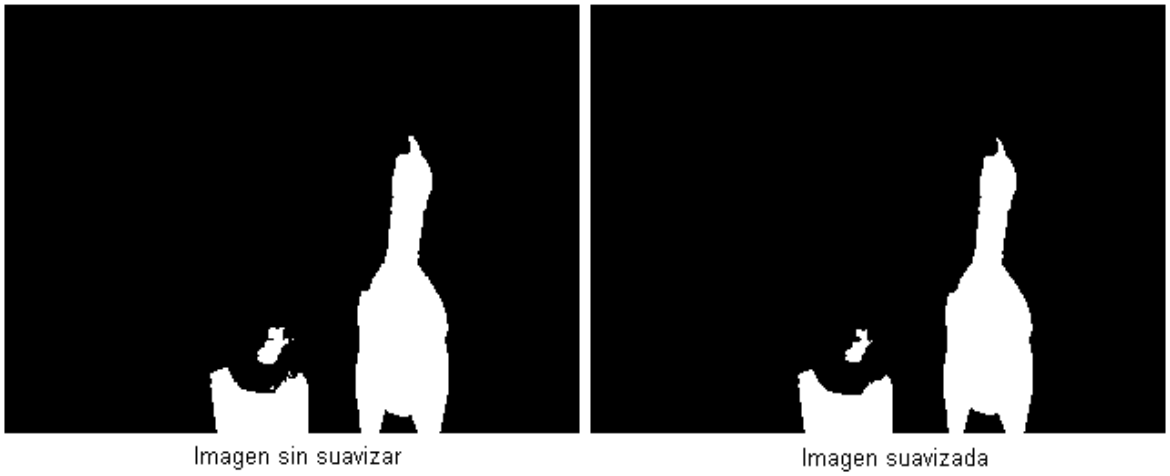


Figura 3.10 Suavizado de bordes

Al hacer el suavizado se desgastan los bordes, incluidos los interiores, de no haber realizado el rellenado algunos objetos pueden llegar a fragmentarse. Se observa cómo han desaparecido las virutas de la parte inferior.

Paso 5: Filtrado por tamaño

Aun realizando todos los procesos anteriores, todavía es posible que queden virutas demasiado grandes y con este proceso se realiza un tamizado por tamaños.

Se realiza con el siguiente comando:

```
BW = bwareaopen(BW, minPixels);
```

En la imagen de ejemplo no han quedado fragmentos lo suficientemente grandes para que este comando haga su efecto. Muestro el ejemplo que aparece en la ayuda de Matlab.

Figura 3.11 Filtrado de objetos pequeños

En esta imagen de muestra se puede ver como las letras pequeñas han desaparecido tras hacer el filtrado por tamaño.

Paso 6: Etiquetado y conteo

Una vez refinada la imagen se procede a su **etiquetado** que asigna a cada bloque de pixeles **conectados** un identificador.

Conectividad de pixeles:

Para determinar si dos píxeles forman parte del mismo bloque se definen dos tipos de conectividad: 4-Conectados y 8-Conectados. Estos criterios se utilizan en varios procesos elementales, como el etiquetado.

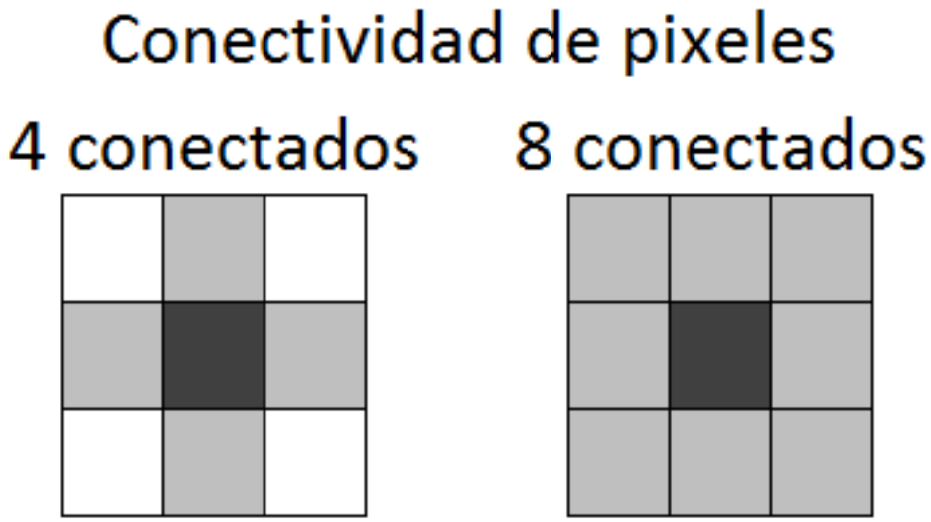


Figura 3.12 Conectividad de pixeles

Labelling / Etiquetado:

Proceso que trabaja sobre una imagen binaria y como resultado obtiene una imagen indexada. En la que cada bloque de píxeles conectados tiene un identificador incremental.

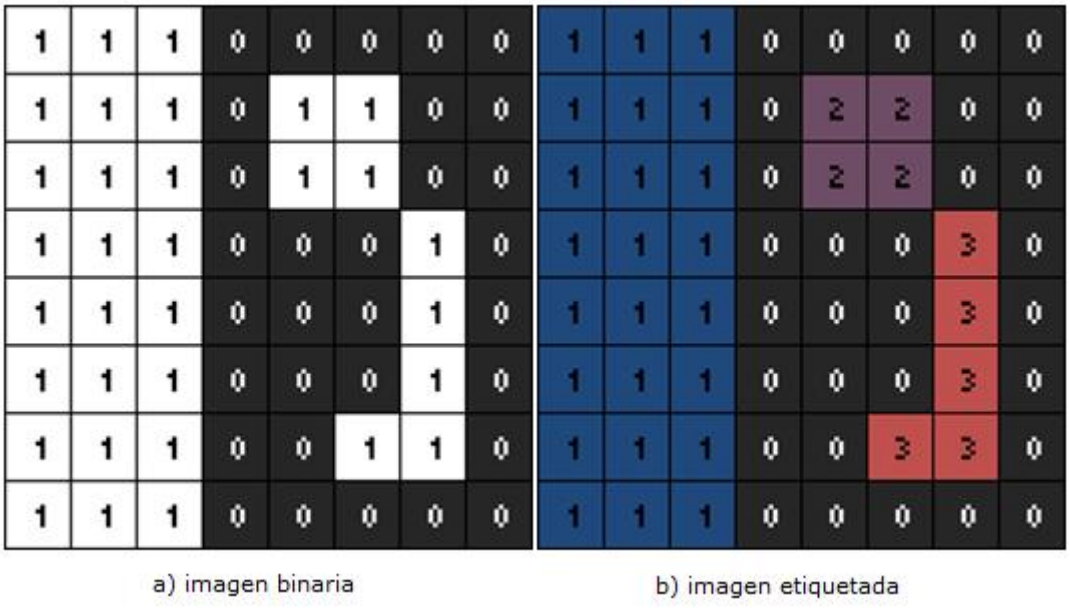


Figura 3.13 Etiquetado de una imagen

En la figura anterior se ve en detalle el proceso de etiquetado, en la matriz etiquetada (b) se puede ver el identificador incremental que se asigna a cada bloque de pixeles. Para visualizar la imagen indexada resultado se aplica una paleta de colores.

En Matlab el etiquetado se realiza con el siguiente comando:

```
L=bwlabel(BW,8); %etiquetado
```

Resultado del proceso:

El proceso de contar los objetos concluye al calcular el máximo valor de las etiquetas y así obtener el valor buscado.

```
objetos = max(max(L)); %conteo (VALOR BUSCADO)
```

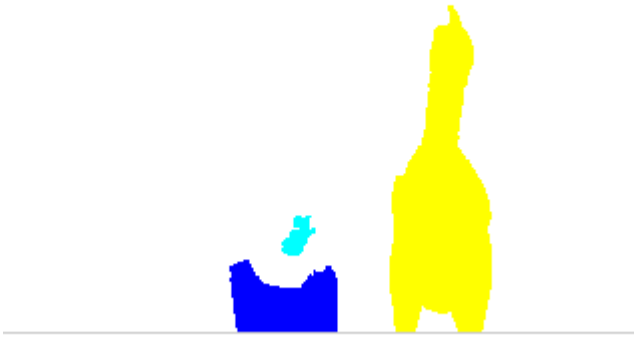


Figura 3.14 Imagen etiquetada resultado del proceso

La respuesta que se obtiene en la imagen de prueba es **3 objetos.**

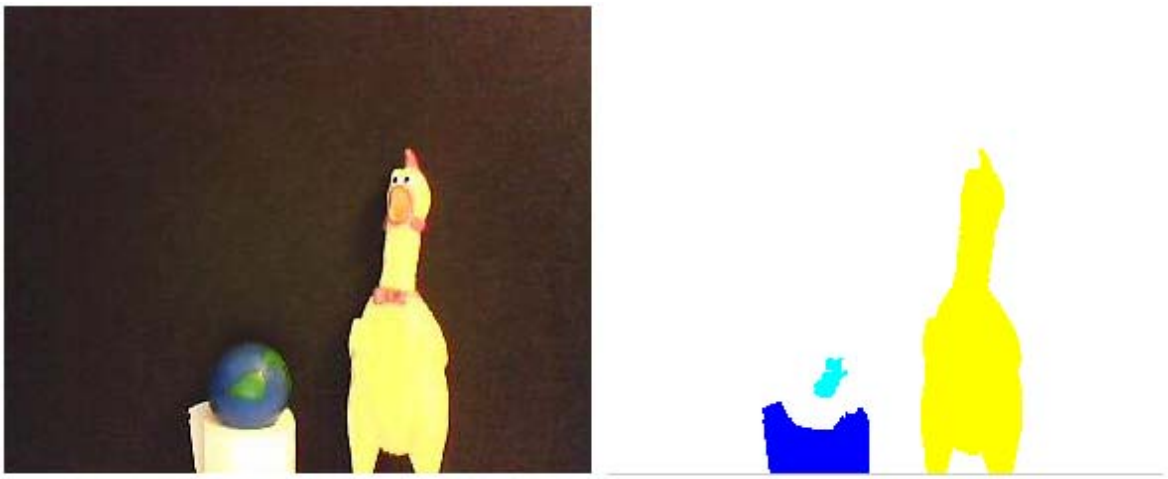


Figura 3.15 Resultado del proceso contador de objetos

En este caso ha habido suerte, se puede ver como durante el proceso han surgido problemas con el reflejo de la pelota de goma por el hecho de estar pegado al rollo de papel y no ofrecer demasiado contraste con el fondo.

A continuación se muestra un ejemplo de la ayuda de Matlab donde se puede apreciar con mayor claridad la utilidad del proceso de contar objetos:

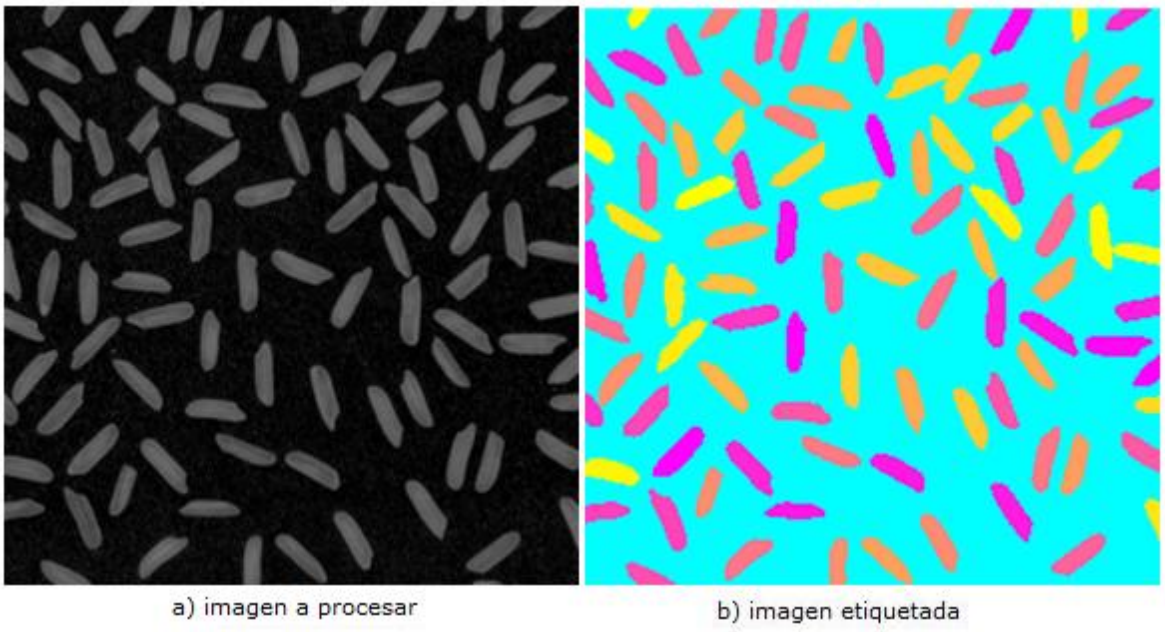


Figura 3.16 Proceso de etiquetado completo

Como se muestra en la figura 3.16 se puede utilizar para el control de calidad de los granos de arroz, de hecho la calidad del arroz está definido por el porcentaje de granos que no están rotos. Solo queda por identificar los granos rotos realizando un proceso posterior.

Conclusiones:

El entorno de desarrollo Matlab es muy avanzado y se logra el control absoluto de todo el proceso resultando ser un entorno excelente para el diseño de algoritmos de procesado de imágenes a parte de las otras ramas que abarca el producto. La ayuda del producto es muy amplia, específica y tiene una comunidad activa que lo soporta.

3.2.2 SIMULINK

Simulink es un entorno de diagramas de bloque para la simulación multidominio y el diseño basado en modelos. Admite el diseño y la simulación a nivel de sistema, la generación automática de código y la prueba y verificación continuas de los sistemas embebidos. Este entorno permite al ingeniero centrarse en la ingeniería y así evitar los problemas derivados de la programación. Se parece mucho a los entornos de simulación de circuitos, estilo **Electronic Workbench**.

Para revisar rápidamente este entorno de desarrollo que viene emparejado con Matlab se va a explicar el proceso del contador de objetos visto en la sección de Matlab.

3.2.2.1 Contador de objetos

Estos son los pasos a realizar en este algoritmo:

Paso 1: Lectura del fichero

En este paso se carga en memoria la imagen a procesar.

Paso 2: Conversión a mapa de bits

Separar los objetos del fondo. Aprovechando el contraste de la imagen.

Paso 3: Rellenar huecos

Recupera pixeles interiores que forman parte del objeto y no se han reconocido.

Paso 4: Suavizado de bordes

Redondear aristas en la imagen para suavizar el contorno de los objetos.

Paso 5: Filtrado por tamaño

Eliminar virutas pequeñas.

Paso 6: Etiquetado y conteo

Contabilizar bloques de pixeles.

En Matlab se ha agrupado la funcionalidad requerida dentro de una función para poderla reutilizar, en Simulink se agrupa en un bloque funcional (**modelo**) que recibe en su entrada la imagen y extrae en su salida el número de objetos.

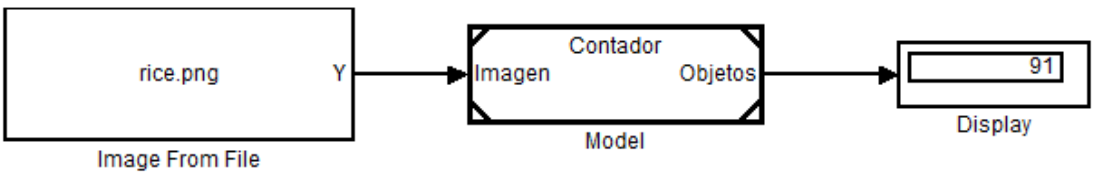


Figura 3.17 Modelo Cliente

En la figura anterior se muestra el **Modelo Cliente** que consume los servicios del **Modelo Contador de Objetos**. Este modelo se encarga del primer paso, leer la imagen de fichero. Se pasa la imagen por el bloque contador y en el Display se visualiza el número de objetos detectados.

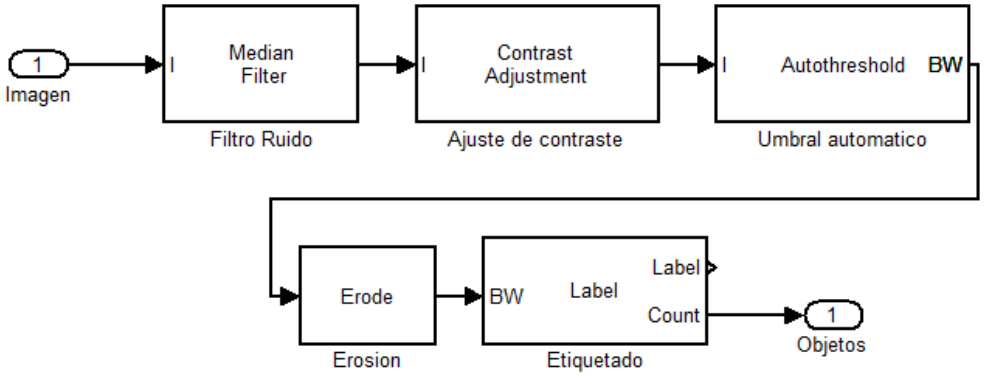


Figura 3.18 Modelo del Contador de Objetos

Esto es lo que contiene el Modelo del Contador de Objetos, es ligeramente diferente a como se implementó en Matlab. La conversión a mapa de bits se realiza en 3 bloques, un filtro mediana para eliminar el ruido de imagen, un contraste automático para aprovechar todo el margen dinámico de la imagen y el umbral automático. En este proceso falta el paso de rellenado de huecos y se ejecuta directamente la erosión y el etiquetado. En este ejemplo se ha utilizado una imagen un poco más difícil que tiene agregado un gradiente de luminosidad. Se han insertado visualizadores para ir viendo los resultados intermedios.

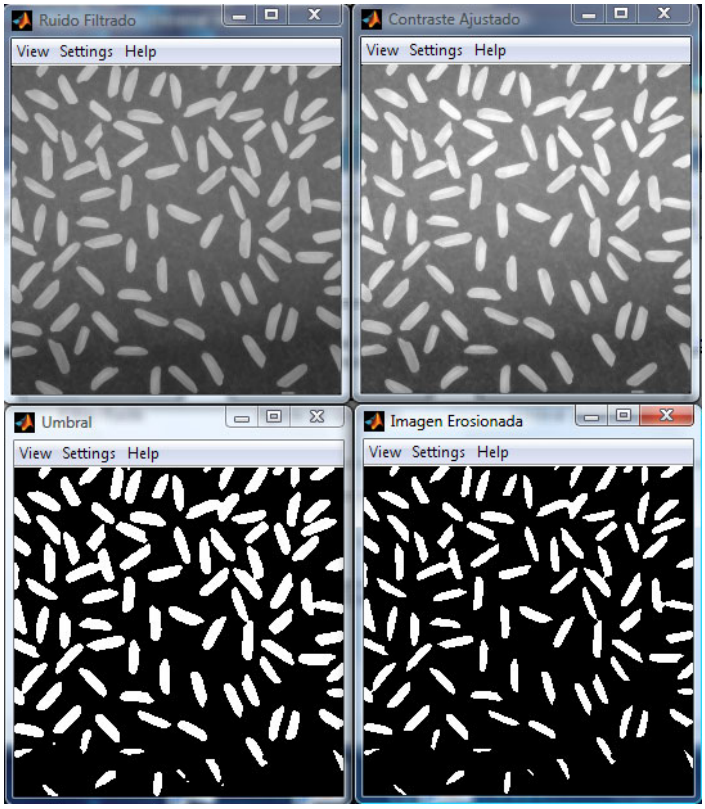


Figura 3.19 Inspecciones durante el proceso

Se puede observar como la imagen utilizada afecta al contador cuando los granos no ofrecen suficiente contraste con el fondo.

En la siguiente imagen se ha realizado el conteo de forma manual y se resaltan los fallos que se pueden encontrar en este tipo de proceso, tanto automático como manual. Los granos 16 y 17, 44 y 45 están demasiado juntos y se pueden confundir con un único objeto. Debajo del grano 20 se me ha escapado uno y tras este el proceso el resultado obtenido es de 96, tardando varios minutos.

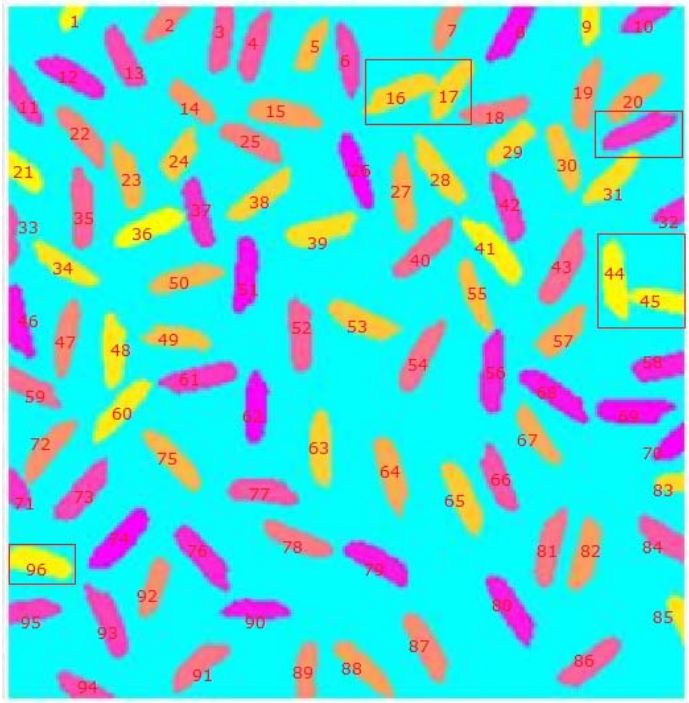


Figura 3.20 Granos de arroz contados de forma manual

Ni los humanos ni las maquinas son infalibles. En todo proceso de medición, tanto automático como manual, se deben considerar los umbrales de error. Los umbrales de error deben ser un requisito definido antes de comenzar con el diseño de un algoritmo de procesamiento de imágenes.

Conclusiones:

Sobre el entorno de desarrollo Simulink se puede decir que es tremendamente avanzado, aunque es complicado lograr el control absoluto de todo el proceso. Al ser un entorno visual permite centrarse en el diseño de algoritmos y evita tener que adentrarse en el código. La ayuda del producto es extensa y tiene una comunidad activa que lo soporta. El inconveniente de los entornos visuales es que se pierde bastante tiempo recolocando elementos y a veces se aglomera resultando ininteligible.

3.3 GENERACIÓN DE CÓDIGO DE PROCESADO DE IMÁGENES

Una vez concluido el diseño, tanto desde Matlab como desde Simulink se puede generar código ejecutable. Matlab tiene el Matlab Compiler que permite generar librerías de enlazado dinámico (.dll en Windows y .so en Linux) y ejecutables. El inconveniente es que para este proyecto es necesario realizar la compilación cruzada y no se ha podido realizar, aparte que requiere tener instalado en el sistema destino un entorno de ejecución (MCR) que ocupa casi tanto como Matlab. Se puede generar código C usando el Real-Time Workshop desde Simulink y se puede configurar el entorno de compilación para hacer compilación cruzada. Se optó por usar esta alternativa y por eso se realizó el modelo en Simulink.

Veamos el diagrama de la generación del código:

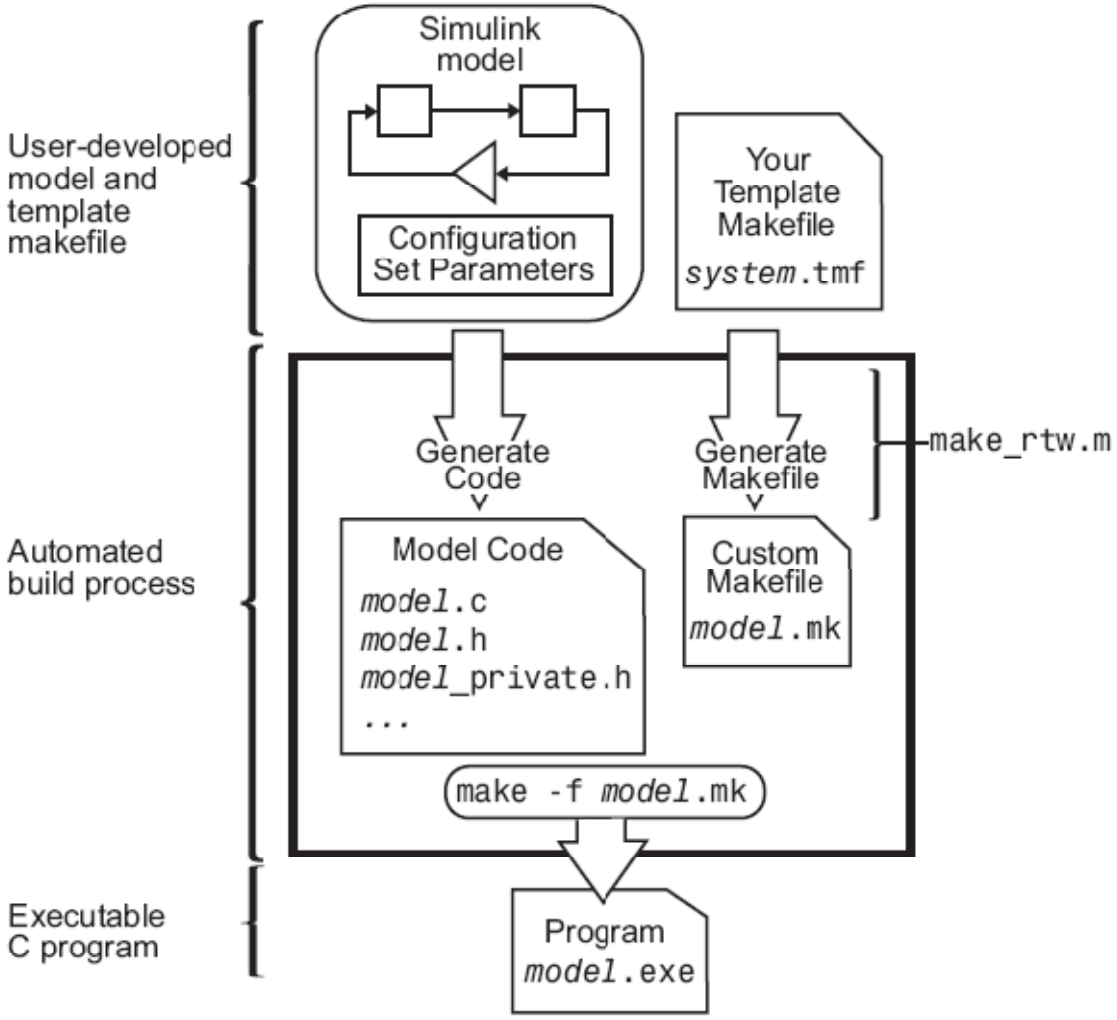


Figura 3.21 Generación de código desde Simulink

Para familiarizarse con Real-Time Workshop es recomendable seguir el tutorial **"First Look at Generated Code"** de la documentación de referencia [12]. El código se ha generado siguiendo los pasos descritos en dicho tutorial.

El código generado está perfectamente documentado tanto con comentarios en línea como en la páginas html de documentación que se generan conjuntamente.

A continuación se muestra el código del bloque Erosión que se ejecuta como un paso intermedio dentro de la función *Contador_output*:

```
    /* S-Function (svipmorphop): '<Root>/Erosion' */
    idxMaxVal = 0;
    i = 0;
    for (m = 0; m < 240; m++) {
      Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal] = true;
      idxMaxVal++;
    }

    for (i_0 = 0; i_0 < 316; i_0++) {
      Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal] = true;
      idxMaxVal++;
      memcpy((void *)(&Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal]), (void *)
             (&Contador_B.Autothreshold[i]), 235U * sizeof(boolean_T));
      idxMaxVal += 235;
      i += 235;
      Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal] = true;
      idxMaxVal++;
      Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal] = true;
      idxMaxVal++;
      Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal] = true;
      idxMaxVal++;
      Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal] = true;
      idxMaxVal++;
    }

    for (i_0 = 317; i_0 < 321; i_0++) {
      for (m = 0; m < 240; m++) {
        Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal] = true;
        idxMaxVal++;
      }
    }

    idxMaxVal = 0;
    i = 0;
    for (i_0 = 0; i_0 < 316; i_0++) {
      for (m = 0; m < 235; m++) {
        Contador_B.Erosion[i] = true;
        for (maxVal = 0; maxVal < Contador_DWork.Erosion_NUMNONZ_DW; maxVal++) {
          if (!Contador_DWork.Erosion_ONE_PAD_IMG_DW[idxMaxVal +
              Contador_DWork.Erosion_ERODE_OFF_DW[maxVal]]) {
            Contador_B.Erosion[i] = false;
            maxVal = Contador_DWork.Erosion_NUMNONZ_DW;
          }
        }

        idxMaxVal++;
        i++;
      }

      idxMaxVal += 5;
    }
```

En la cabecera del modelo contador (Contador.h) se encuentran los puntos de entrada:

```
     /* Model entry point functions */
     extern void Contador_initialize(boolean_T firstTime);
     extern void Contador_output(int_T tid);
     extern void Contador_update(int_T tid);
     extern void Contador_terminate(void);
```

Conclusiones:

El código generado no cumple con los requisitos del caso abordado. Es un programa en tiempo real listo para ejecutarse de forma continua en el sistema embebido. En el caso abordado se requiere una ejecución bajo demanda. La alternativa consiste en implementar el proceso diseñado con Matlab o Simulink utilizando librerías de procesamiento de imágenes.

3.4 LIBRERIAS DE PROCESAMIENTO DE IMÁGENES

Hay cientos de librerías para realizar procesamiento de imágenes, ver referencia [14], muchas de ellas están especializadas en tareas concretas y en cambio otras tratan asuntos generales. Las hay con licencias propietarias y de código abierto. Una de las más famosas de código abierto es OpenCV, fue utilizada en el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford, el ganador en el año 2005 del Gran desafío DARPA.

Se han analizado varias librerías de procesado de imágenes de código abierto:

Camellia, cips2ed, IPL98, OpenCV y QDBMP

Salvo QDBMP todas han dado numerosos problemas de compilación o eran incompatibles con el software de la cámara por estar escritas en C++, solo mencionaré OpenCV por su alto grado de aceptación.

3.4.1 OPENCV

OpenCV es una biblioteca libre de visión artificial inicialmente desarrollada por Intel. Desde su primera versión se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esta gran aceptación se debe a que su publicación con licencia BSD. Se permite su uso libre para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV está escrita en código C y C++ optimizados y es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estérea y visión robótica.

No se ha podido utilizar esta librería por problemas de espacio y relacionados con la compilación cruzada de C++ y la librería estándar de C presente en la cámara (uClibc). Se detallarán al final del capítulo.

3.4.2 QDBMP

Para salvar las restricciones de espacio en la cámara se ha optado por utilizar una librería reducida capaz de leer el formato BMP empleado en las instantáneas de la cámara.

QDBMP, ver referencia [15], es una librería de C minimalista y multi-plataforma para el manejo de imágenes BMP. La librería es extremadamente ligera, consiste únicamente en dos pequeños ficheros y solo tiene dependencias con la librería estándar de C. Está concebida para proporcionar capacidades básicas de manejo de imágenes a pequeñas aplicaciones utilizando el ampliamente usado formato BMP. Es libre y de código abierto, distribuida bajo la licencia MIT.

La librería soporta las siguientes variantes del formato BMP:

- 32 BPP (Bits Por Pixel) sin compresión (los valores alpha se ignoran)
- **24 BPP sin compresión**
- 8 BPP sin compresión (color indexado)

Como se ha descrito en el apartado 2.2.1 la profundidad de color más utilizada es la de 24bpp (8 bits por cada canal de color RGB), con esta profundidad de color se pueden representar 255 niveles de intensidad en cada color produciendo un total de 16.581.375 colores.

Veamos los métodos originales de la librería:

```
/* Construction/destruction */
BMP*   BMP_Create ( UINT width, UINT height, USHORT depth );
void   BMP_Free   ( BMP* bmp );

/* I/O */
BMP*   BMP_ReadFile  ( const char* filename );
void   BMP_WriteFile ( BMP* bmp, const char* filename );

/* Meta info */
UINT   BMP_GetWidth  ( BMP* bmp );
UINT   BMP_GetHeight ( BMP* bmp );
USHORT BMP_GetDepth  ( BMP* bmp );

/* Pixel access */
void   BMP_GetPixelRGB   ( BMP* bmp, UINT x, UINT y, UCHAR* r, UCHAR* g, UCHAR* b );
void   BMP_SetPixelRGB   ( BMP* bmp, UINT x, UINT y, UCHAR r, UCHAR g, UCHAR b );
void   BMP_GetPixelIndex ( BMP* bmp, UINT x, UINT y, UCHAR* val );
void   BMP_SetPixelIndex ( BMP* bmp, UINT x, UINT y, UCHAR val );

/* Palette handling */
void BMP_GetPaletteColor ( BMP* bmp, UCHAR paleteIndex, UCHAR* r, UCHAR* g, UCHAR* b );
void BMP_SetPaletteColor ( BMP* bmp, UCHAR paleteIndex, UCHAR r, UCHAR g, UCHAR b );

/* Error handling */
BMP_STATUS  BMP_GetError ();
const char* BMP_GetErrorDescription ();
```

El código de la librería tiene un estilo de programación Orientada a Objetos que ha resultado ser bastante didáctico y claro. Ofrece un método para obtener los colores RGB de un pixel según sus coordenadas XY.

Analicemos dicho método:

```
/*************************************************************
 Populates the arguments with the specified pixel's RGB
 values.
*************************************************************/
void BMP_GetPixelRGB( BMP* bmp, UINT x, UINT y, UCHAR* r, UCHAR* g, UCHAR* b )
{
  UCHAR* pixel;
  UINT   bytes_per_row;
  UCHAR  bytes_per_pixel;

  if ( bmp == NULL || x < 0 || x >= bmp->Header.Width || y < 0 || y >= bmp->Header.Height )
  {
    BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
  }
  else
  {
    BMP_LAST_ERROR_CODE = BMP_OK;

    bytes_per_pixel = bmp->Header.BitsPerPixel >> 3;

    /* Row's size is rounded up to the next multiple of 4 bytes */
    bytes_per_row = bmp->Header.ImageDataSize / bmp->Header.Height;

    /* Calculate the location of the relevant pixel (rows are flipped) */
    pixel = bmp->Data + ( ( bmp->Header.Height - y - 1 ) * bytes_per_row + x *
bytes_per_pixel );


    /* In indexed color mode the pixel's value is an index within the palette */
    if ( bmp->Header.BitsPerPixel == 8 )
    {
      pixel = bmp->Palette + *pixel * 4;
    }

    /* Note: colors are stored in BGR order */
    if ( r )   *r = *( pixel + 2 );
    if ( g )   *g = *( pixel + 1 );
    if ( b )   *b = *( pixel + 0 );
  }
}
```

Hacer un procesado usando este método puede resultar altamente ineficiente debido a los cálculos redundantes que se repiten al acceder a cada pixel. Se ha optado por extender esta librería con las funciones requeridas.

3.4.2.1 EXTENSIÓN REALIZADA A QDBMP

La librería QDBMP se ha extendido para cumplir con el caso práctico que se plantea en el capítulo 6. A efectos introductorios solo hay que saber que es requisito contar el número de píxeles que tienen un color determinado en la imagen capturada por la cámara.

La extensión realizada a QDBMP se considera de bajo nivel por haber tenido que combinar los algoritmos de procesado con la lógica del formato BMP y se ha tenido en cuenta el entorno de ejecución, a cambio se ha ganado velocidad de ejecución y se han ahorrado recursos.

Teniendo en cuenta que la imagen es un fichero temporal y que en la cámara se guarda en un ram-disk, ya está en memoria, no tiene mucho sentido crear un método para extraer el array de pixeles a otra región de memoria. Hay que destacar que la imagen capturada en la cámara ocupa 900 MB y copiar los pixeles es una tarea costosa en tiempo y recursos. En su lugar

se ha implementado el procesado sobre el propio formato BMP, con las implicaciones que eso tiene, por ejemplo las filas de la imagen están invertidas y el color está almacenado en orden BGR. El gran ahorro de recursos y aumento de velocidad se ha conseguido dejando el resultado del algoritmo en el mismo espacio de memoria (en algunos casos).

No son técnicas de programación recomendadas pero resultan un mal necesario, hubo que medir los tiempos para optimizar el proceso porque tardaba demasiado. La misma imagen procesada en la máquina virtual tarda 60ms, en la cámara tarda 37.684ms, **¡37 segundos!** Tras una serie de mejoras se ha quedado en 150ms, **¡251 veces más rápido!**

Métodos agregados:

```
/* Converts RGB to grayscale (Y = 0.299*R + 0.587*G + 0.114*B) */
BMP* BMP_ConvertToGray  ( BMP* source );

/* Apply Quick Edge mask filter */
void BMP_QuickEdge ( BMP* bmp );

/* Color detection and count */
BMP* BMP_DetectColor ( BMP* source, UCHAR r, UCHAR g, UCHAR b, UCHAR threshold );
BMP* BMP_DetectColorProp ( BMP* source, UCHAR r, UCHAR g, UCHAR b, UCHAR threshold );
UINT BMP_CountColorProp  ( BMP* source, UCHAR r, UCHAR g, UCHAR b,
                           UCHAR threshold, UINT stop, UCHAR average, UCHAR velocity );
UINT BMP_FastCountColorProp ( BMP* source, UCHAR r, UCHAR g, UCHAR b, UCHAR threshold);
UINT BMP_VVCountColorProp   ( BMP* source, UCHAR r, UCHAR g, UCHAR b, UCHAR threshold);
```

- ***BMP_ConvertToGray***: Convierte la imagen origen a escala de grises usando los coeficientes definidos por el ITU-R, crea el resultado en un nuevo espacio de memoria. Lo notable en esta función es que cuando trabaja con una imagen indexada solo es necesario trabajar con la paleta de colores. El resultado siempre queda como imagen indexada. Hay que recordar que casi todos los algoritmos de procesado de imágenes trabajan sobre imágenes en escala de grises por eso esta función es importante.
- ***BMP_QuickEdge***: Aplica un filtro de mascara (kernel de combolución) sobre una imagen en escala de grises. Concretamente utiliza el filtro Quick Edge. Este filtro encuentra los bordes de los objetos en una sola pasada del kernel de combolución al contrario que otros filtros que requieren de 4 a 8 pasadas con diferentes kernels. Trabaja en memoria sobre la imagen. Es decir, transforma la imagen sin crear una nueva. El Kernel de combolución es el siguiente:

```
/*quick_edge_filter_mask*/
short quick_mask[3][3] =  {
        {-1,  0, -1},
        { 0,  4,  0},
        {-1,  0, -1} };
```

-1	0	-1
0	4	0
-1	0	-1

Figura 3.22 Máscara del filtro Quick Edge

La única reseña a este filtro aparece en la referencia [19] y dice:

The "quick mask" is so named because it can detect edges in all eight directions in one convolution. This has obvious speed advantages... The results of the quick mask are as good as the other masks, and it operates in one-eighth the time.

Este es el resultado de aplicar el algoritmo:

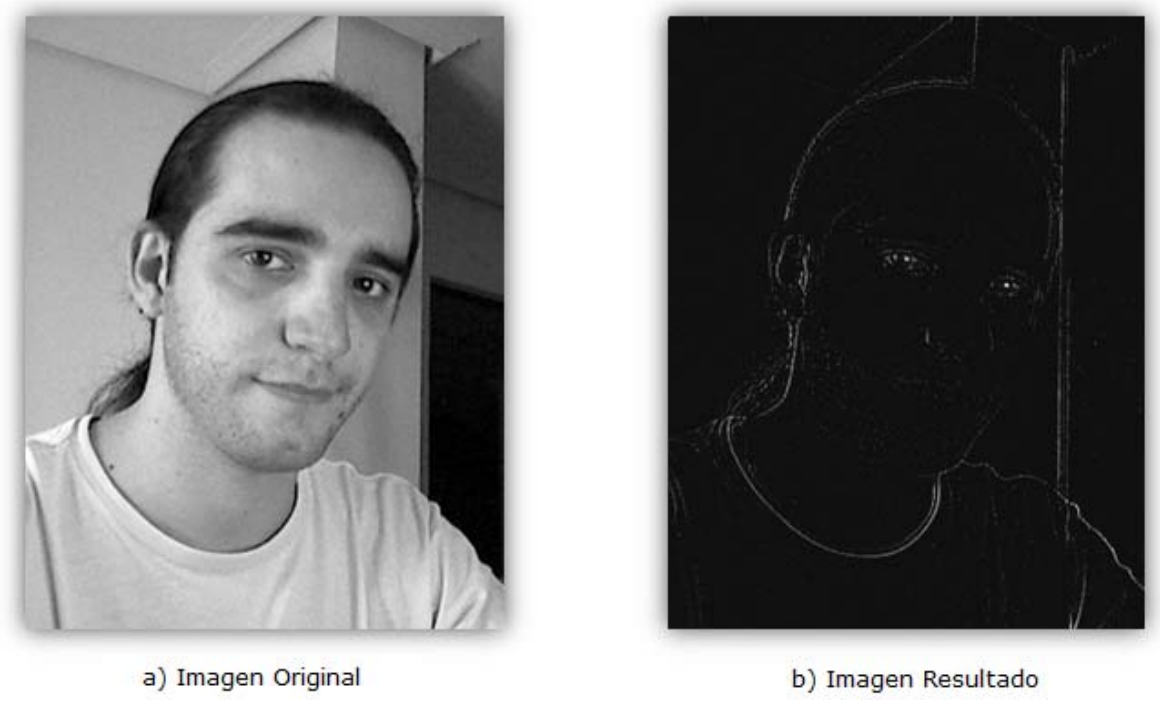


Figura 3.23 Aplicación del filtro Quick Edge

Este filtro localiza los bordes de los objetos. Los filtros de detección de bordes tienen muchas aplicaciones en el campo de la visión por computador. Este filtro en concreto emplea menos tiempo en su ejecución. Además, se ha implementado la combolución sobre la propia imagen para reducir a la mitad la memoria requerida. Los detalles de esta implementación se describen en el siguiente diagrama.

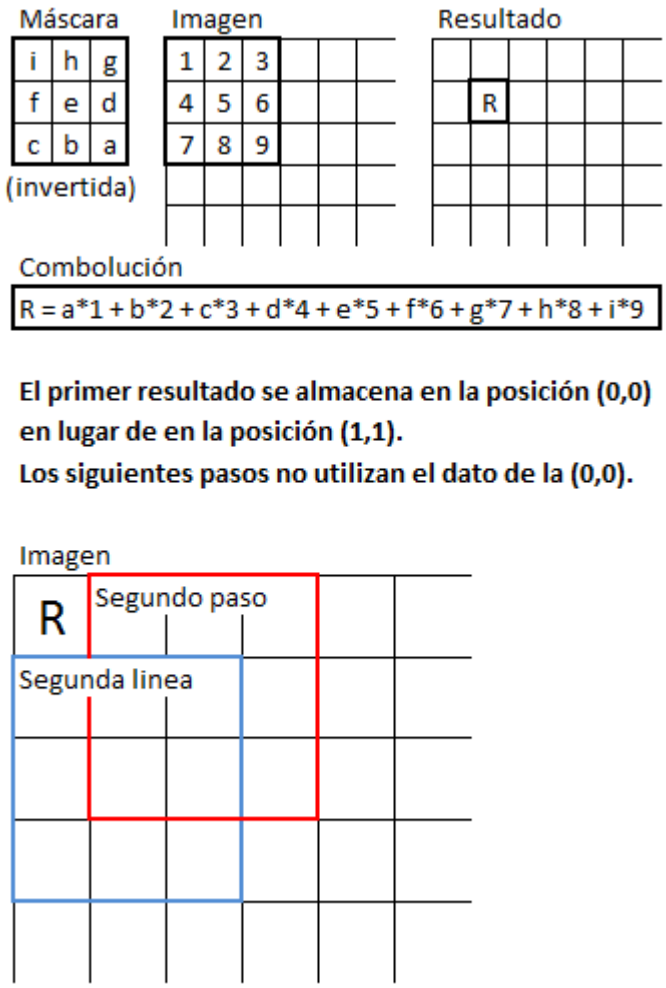


Figura 3.24 Explicación de sobrescritura (0,0)

En la figura anterior se ve como se sobrescribe la posición (0,0) con el dato del primer cálculo al aplicar el kernel de combolución. Al procesar toda la imagen introduce una translación (-1,-1) sobre la imagen. En el resultado no se aprecia pero hay que tenerlo presente si se realizan procesos a continuación.

- ***BMP_DetectColor***: Detecta un color especificado con un margen de tolerancia (T). Tiene problemas con los cambios de luminosidad. Crea una imagen con solo los pixeles encontrados sobre fondo negro.

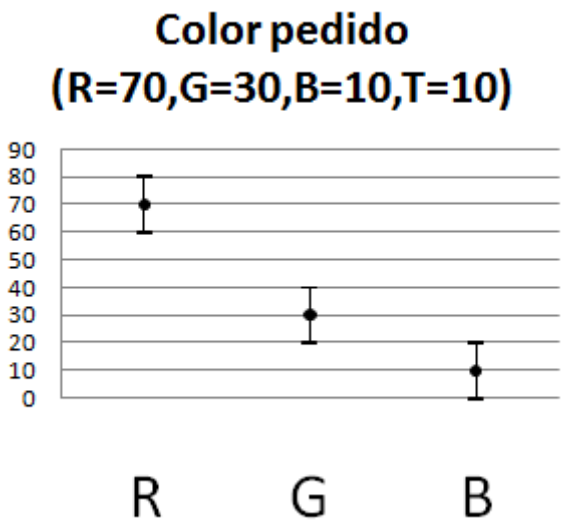


Figura 3.25 Detección de color por tolerancia

Equivale a un cubo de radio (T) sobre el color pedido en el espacio de colores RGB. El método realiza un promediado del color (sum{*Color*}) de un vecindario 3x3 para evitar ruidos:

```
if (abs(sumRed   - r) < threshold &&
    abs(sumGreen - g) < threshold &&
    abs(sumBlue  - b) < threshold)
```

- ***BMP_DetectColorProp***: Detecta un color especificado como una proporción. El resultado es una imagen con los pixeles que cumplen el criterio sobre fondo negro.

Tras analizar los problemas del método anterior con las variaciones de luminosidad se decidió diseñar un método de detección. Para que haya un **color definido** distinto de la gama de grises alguna de las componentes debe destacar sobre las otras. Los cambios de luminosidad modifican las tres componentes por igual. El color pedido (R=70,G=30,B=10) en todo el rango de luminosidad equivale a (R=70+x,G=30+x,B=10+x) donde la x es la alteración de luminosidad entre luces y sombras del objeto. El margen de tolerancia permite ligeros cambios de color al igual que el método anterior. Bajo estos criterios tenemos un sistema de 3 inecuaciones:

```
R - Margen < x + RojoCriterio  < R + Margen
G - Margen < x + VerdeCriterio < G + Margen
B - Margen < x + AzulCriterio  < B + Margen
```

En su implementación ni siquiera hay que resolver el sistema, basta con verificar que tiene solución. En cada inecuación el color criterio pasa restando a ambos lados y las 6 condiciones que se deben cumplir se sacan por igualación de cada lado con sus dos opuestos. También trabaja sobre el color promedio (sum{*Color*}) de un vecindario 3x3.

```
if (sumRed   - r - threshold < sumGreen - g + threshold &&
    sumRed   - r - threshold < sumBlue  - b + threshold &&
    sumGreen - g - threshold < sumRed   - r + threshold &&
    sumGreen - g - threshold < sumBlue  - b + threshold &&
    sumBlue  - b - threshold < sumRed   - r + threshold &&
    sumBlue  - b - threshold < sumGreen - g + threshold)
```

Se puede verificar de otra forma averiguando cual es el color menos intenso, restarle su intensidad a los otros 2 componentes y ver si cumplen el criterio. El sistema de inecuaciones resulta más apropiado aunque es más difícil de comprender.

La forma del criterio en el espacio de colores RGB es casi igual de complejo. Es un paralelepípedo que apunta en la dirección del color criterio. Se ve mejor con un ejemplo concreto.

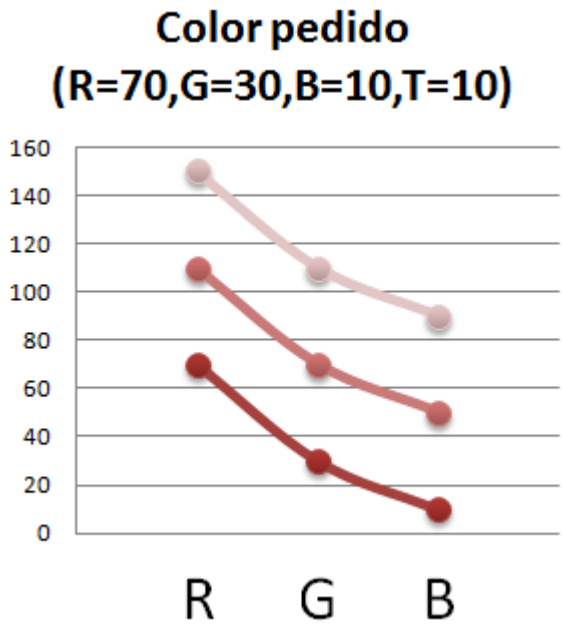


Figura 3.26 Detección de color por proporción

Este ejemplo corresponde solo a una gama de luminancia. Hay que tener en cuenta que también se detectan otras gamas correspondientes a los cambios de color dentro de la tolerancia.

A continuación se puede ver una comparativa entre los distintos métodos:

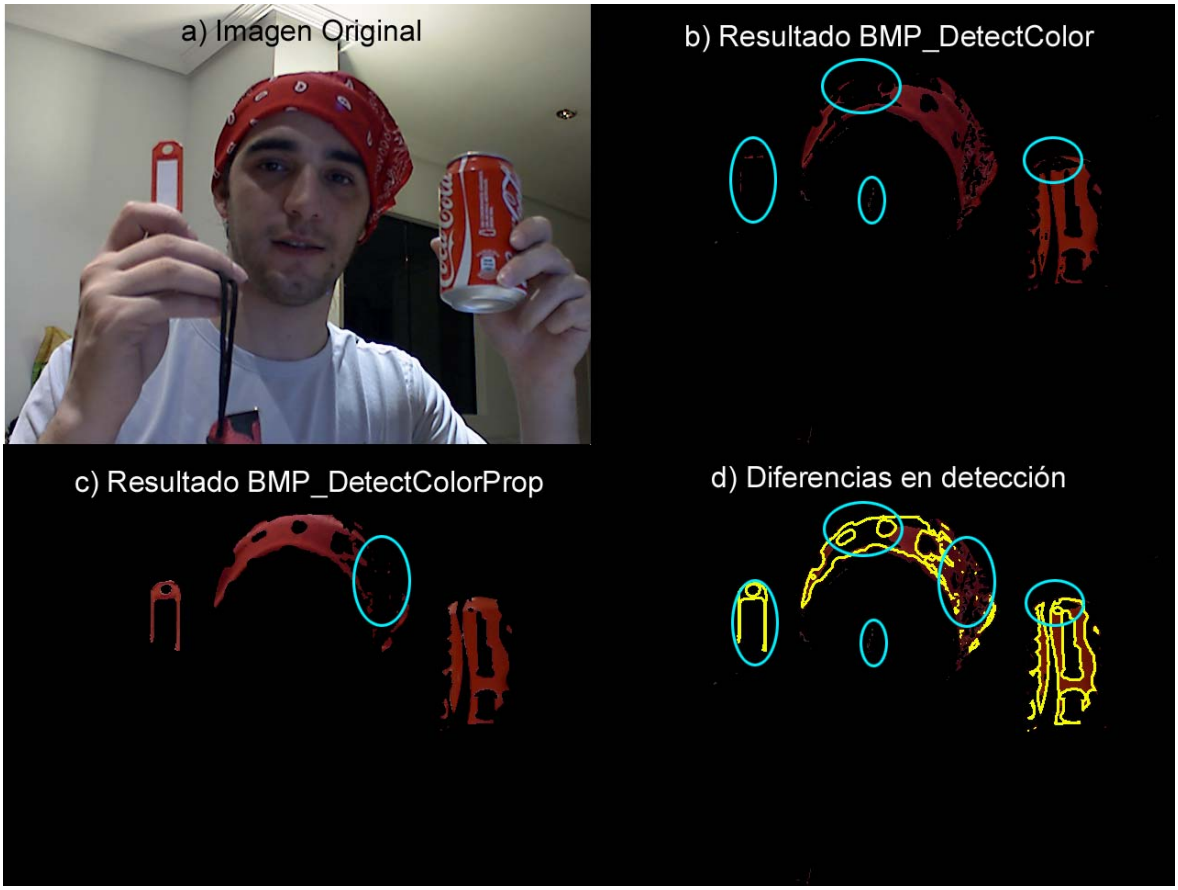


Figura 3.27 Comparativa de métodos de detección del color

Se puede apreciar en la Figura 3.27 cómo el método ***BMP_DetectColor*** tiene dificultades con los brillos, prácticamente no encuentra el llavero y da algunos falsos positivos en la nariz, oreja y mano derecha. El método ***BMP_DetectColorProp*** lo hace bastante mejor aunque sigue fallando en las sombras. De la mano derecha cuelga otro llavero rosa que ningún método lo confunde.

- ***BMP_CountColorProp***: Igual que el anterior pero ahora solo devuelve el numero de pixeles encontrados en lugar de crear una imagen con el resultado. Permite desactivar el filtro de ruidos average 3x3. Esto acelera x9 la velocidad de procesado. Admite un parámetro velocidad para saltar pixeles y estimar el resultado. Admite un parámetro stop para detener el procesado cuando se alcanza la cantidad especificada de pixeles encontrados.

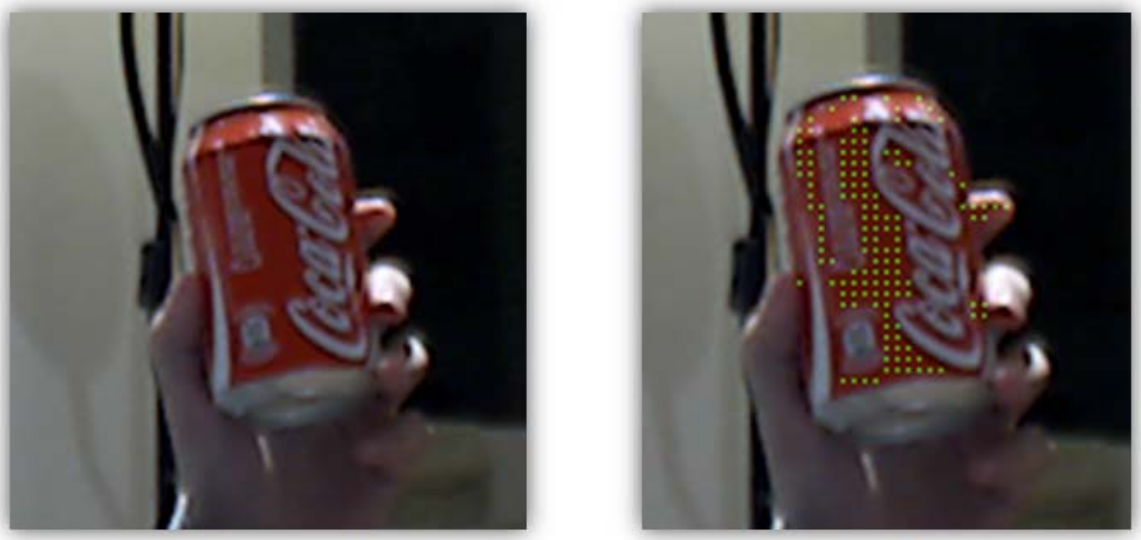
Este método contiene todas las opciones que permiten acelerar el procesado. Una velocidad 3 hace que solo se procese uno de cada 3 píxeles en cada fila y una de cada 3 filas. Utilizar la velocidad superior a 1 produce resultados estadísticos basados en la muestra. Se asume que, al cumplir el criterio un pixel, está rodeado en su vecindad NxN de pixeles que también cumplen el criterio. Por ello el resultado de cada coincidencia se multiplica por el peso del pixel.

Para calcular el peso de cada pixel se utiliza la velocidad al cuadrado = V*V. Si se utiliza la velocidad 3 junto con el filtro average 3x3 todos los pixeles de la imagen forman parte del resultado.

- ***BMP_VVCountColorProp***: Igual que el anterior pero ahora la velocidad es variable, aumenta cuando no encuentra pixeles (hasta un límite) y disminuye cuando encuentra. No realiza average ni detiene el proceso al llegar a una cantidad.
- ***BMP_FastCountColorProp***: Igual que el anterior pero se ha fijado una velocidad optimizada que da buenos resultados para las imágenes de la cámara. Al tener una velocidad constante el proceso siempre tarda lo mismo.

Resultados Obtenidos:

En las pruebas realizadas el mejor de todos los procesos de recuento es el ***BMP_FastCountColorProp*** por ser isócrono y devolver un número de pixeles muy bien aproximado al real, además de no depender de cambios de luminosidad. Para depuración se ha coloreado el pixel contado incrementado la intensidad de la componente verde al máximo. Como sabemos, el resultado del proceso no es la imagen en sí, sino el número de pixeles del color indicado.



a) Imagen Original b) Imagen en Depuración

Figura 3.28 Resultado en depuración del recuento BMP_FastCountColorProp

Conclusiones:

Procesar imágenes en un sistema embebido es lento. Al menos con la tecnología empleada solo se puede utilizar en casos muy concretos. Se pueden realizar procesos de visión artificial básicos.

3.5 PROBLEMAS ENCONTRADOS

En el diseño de algoritmos de procesado surgen bastantes dificultades técnicas. Saltarse un paso de rellenado de huecos puede originar problemas al realizar el suavizado porque se desgasten los bordes, incluidos los interiores y algunos objetos pueden llegar a fragmentarse. Lo más grave es que estos fallos no se detecten hasta encontrarse con un caso apropiado. Para evitar este problema hay que trabajar con una base de imágenes de prueba y otra de contraste. En internet se pueden encontrar catálogos de imágenes para pruebas. Y como parte del proceso de diseño es imprescindible construirse una base robusta con suficientes casos positivos, casos negativos, ocultamientos, diversos colores de piel, personas con gafas, con barba, etc. Durante el proceso de diseño se puede caer en casos de overfitting, ajustando parámetros se realiza un "sobreajuste" para cumplir con los casos de la base de entrenamiento, sobre todo si se trabaja con redes neuronales. Para evitar el problema del sobreajuste es necesario reservar un conjunto de imágenes de prueba para realizar un contraste.

Es sorprendente lo fácil que vemos los humanos y lo difícil que es enseñarle a una máquina a realizar tareas similares. Es necesario destacar las dificultades encontradas con la luz y los colores. La luz de una bombilla no es blanca. Ni siquiera la luz del sol es blanca. Las sombras tampoco son negras, dependen de la luz ambiental. Si el cielo está despejado la sombra de un objeto será del color reflejado por el cielo. El ajuste de blancos automático ayuda en estos casos. Y aunque parezca broma, en la oscuridad todos los gatos son pardos, si falta iluminación es muy difícil distinguir los colores. No todos los objetos tienen brillo mate, eso significa que reflejan parte de los colores del entorno, y aunque tengan brillo mate irradian su color a los objetos de alrededor. Los objetos transparentes y los de brillo metálico son un mundo aparte con referencia al color. Una mesa marrón ya no será tan marrón si refleja una pared azul.

Ni los humanos ni las maquinas son infalibles, en todo proceso de medición, tanto automático como manual, se deben considerar los umbrales de error en la fase de diseño.

El mundo de las aplicaciones embebidas tampoco es un camino de rosas. Ya se han destacado algunos problemas en el capítulo anterior con la librería uclibc versión 0.9.27-15 presente en la cámara y su incompatibilidad entre versiones de la misma.

En la generación de código con Matlab no se permite la compilación cruzada y requiere un entorno de ejecución que ocupa casi como Matlab. En Simulink resulta difícil parametrizar los bloques y dejar trazas de ejecución.

Otra tarea laboriosa ha sido la depuración de los algoritmos en la propia cámara. Es necesario crear varios scripts y programas para ejecutar los procesos en la cámara sobre imágenes precargadas y recuperar las imágenes resultado vía ftp. La medición de tiempos de procesado se ha podido realizar utilizando la consola remota. Ha sido muy complejo tener que

integrar el procesado de imagen con la lectura del formato BMP por problemas de rendimiento y recursos.

Aun habiendo sorteado todos los obstáculos sigue siendo un proceso muy lento. El caso práctico que se expone en el capítulo 6 solo se puede implantar en un entorno productivo con pocas exigencias de velocidad. Si se requiere velocidad extrema en un sistema embebido no se puede estar pensando en reutilizar las cámaras de vigilancia. Para ese tipo de procesos mejor hacerlo con un sistema FPGA dedicado, que por cierto se llevan muy bien con Simulink. Ver la referencia [20]. Si la velocidad no es extrema pero el algoritmo es muy complejo usar OpenCV en un servidor dedicado como los vehículos no tripulados del Gran desafío DARPA.

4 ARQUITECTURAS SOA

En este apartado se explica cómo explotar la información generada tras el procesamiento de imágenes mediante un servicio web. Qué tecnologías de servicios web hay para dispositivos de recursos limitados y como generar la estructura de código para implementar un servicio.

4.1 INTRODUCCIÓN

La programación orientada a objetos (OOP – Object-Oriented Programming) introduce una serie de conceptos, como la abstracción y el encapsulamiento. La Programación Basada en Componentes (CBD – Component-Based Development) establece un marco contractual mediante un lenguaje de definición de interfaces entre componentes. La Arquitectira Orientada a Servicios (SOA – Service Oriented Arquitecture) es un paradigma tecnológico para el diseño de aplicaciones basada en servicios. Bajo esta arquitectura las aplicaciones son colecciones estructuradas de servicios individuales. De modo similar a como se construyen los circuitos impresos que interconectan componentes hardware. El propósito de SOA es permitir la fácil cooperación entre un gran número de computadoras conectadas en red.

Hoy en día su implementación más extendida se basa en **Servicios Web** definidos mediante el Lenguaje de Descripción de Servicios Web (WSDL) y soportado por el Protocolo de Transferencia de HiperTexto (HTTP) a través del cual se intercambian mensajes (SOAP), pero no es la única implementación posible. Esta implementación permite sortear los problemas encontrados con los Firewall en aplicaciones CORBA y así tener la capacidad de desarrollar aplicaciones distribuidas a escala mundial independientes de la plataforma.

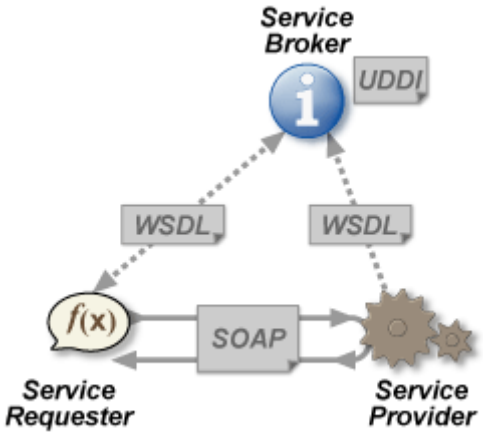


Figura 4.1 Arquitectura de Servicios Web

4.2 DPWS

El problema de implementar Web Services en dispositivos con recursos limitados se aborda mediante (DPWS – Devices Profile for Web Services) que define el juego de restricciones de implementación mínimas para permitir el intercambio seguro de mensajes, descubrimiento, descripción y eventos en estos dispositivos. Ha sido llamado "el USB de Ethernet" porque permite conectar un dispositivo a la red, descubrirlo y utilizar sus servicios desde cualquier otro dispositivo conectado a la misma red. Tiene unos objetivos similares al Universal Plug and Play (UPnP) pero, como añadido, DPWS está completamente alineado con la tecnología de Servicios Web e incluye numerosos puntos de extensión permitiendo la integración sin fisuras de servicios proporcionados por dispositivos en aplicaciones empresariales.

El estándar DPWS fue desarrollado para dotar a los dispositivos de recursos limitados con capacidades de servicios web seguros. Ofrece intercambio seguro de mensajes con servicios web, el descubrimiento dinámico y descripción de servicios Web, suscripción y recepción de eventos de un servicio Web.

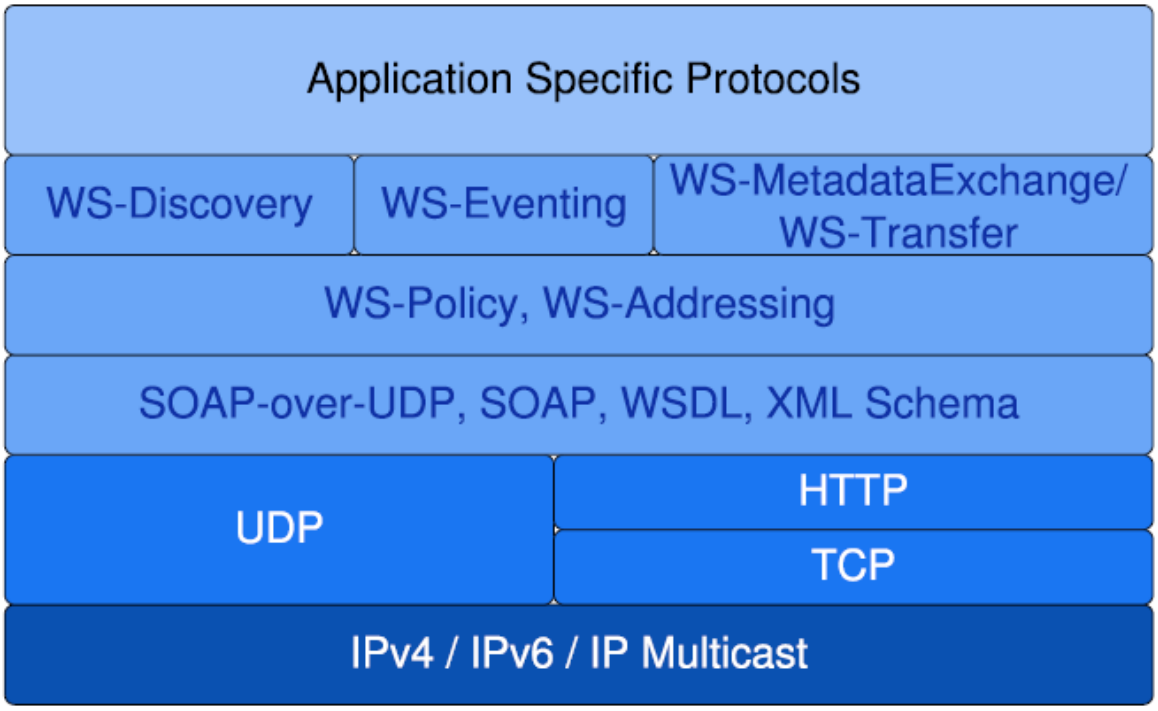


Figura 4.2 Torre de protocolos DPWS

Como se muestra en la figura anterior, DPWS se basa en protocolos bien conocidos y varias especificaciones de servicios Web. Emplea mecanismos de mensajería similares a la arquitectura de servicios web (WSA), con restricciones a la complejidad y al tamaño del mensaje. Se apoya en los protocolos de comunicación IP punto a punto y multidifusión, TCP y HTTP; Utiliza SOAP sobre UDP, SOAP y XML Schema para el intercambio de mensajes. WS-Policy y WS-Addressing se encuentran en la parte superior de la capa de mensajería. WS-Policy se utiliza para

intercambiar y negociar las políticas y los parámetros necesarios para el uso del servicio. WS-Addressing separa la capa de mensajería SOAP de su estrecha unión con HTTP como se define en la especificación SOAP. Se introduce el concepto de encabezados de información de mensajes y referencias de punto final que hacen los usuarios del servicio, proveedores y mensajes transmitidos sean identificables y enrutables.

DPWS define un rol de cliente que utiliza las características que se describen en los siguientes párrafos y un rol de dispositivo que implementa dichas características:

- **Direccionamiento:** DPWS asume que el dispositivo ha obtenido una dirección IP válida mediante DHCP o autoconfiguración IPv6. Normalizado en el estándar WS-Addressing.
- **Descubrimiento:** Este mecanismo permite que el dispositivo se anuncie en la red local con mensajes de multidifusión. Los clientes pueden escuchar estos mensajes o enviar mensajes para buscar dispositivos en la red. Normalizado en el estándar WS-Discovery.
- **Descripción:** Mecanismo que permite la descripción dinámica de los metadatos del dispositivo como los servicios suministrados, información del dispositivo, información del modelo o descripción del servicio. Normalizado en el estándar WS-MetadataExchange.
- **Control y Eventos:** DPWS ofrece servicios con operaciones (similares a las operaciones SOAP de servicios Web) y eventos que actúan como operaciones inversas en las que el intercambio de mensajes inicia en el dispositivo. Lleva asociado un mecanismo de suscripción a eventos. Para transportar datos binarios y otros formatos no XML, DPWS incluye el mecanismo de adjuntos MTOM. Los eventos están descritos en el estándar WS-Eventing.
- **Presentación:** Es una vía alternativa para obtener información sobre el dispositivo y controlarlo. Es informado como parte de la descripción del dispositivo como un IRI HTTP que puede usarse en un navegador normal y corriente.
- **Seguridad:** DPWS define los conceptos de perfiles de seguridad. En la especificación se define uno de ellos. Como los requisitos de seguridad son muy dependientes de los escenarios de aplicación, el perfil de seguridad definido se orienta a la implementación simple y ligera más que a cubrir muchos de los posibles requisitos de seguridad. Consta de dos características principales: descubrimiento y canales seguros.
- **Extensibilidad y composibilidad:** Una de las grandes ventajas de DPWS es la natural facilidad de extensión y composición de su especificación y los servicios web. Los servicios web definen mayormente interfaces y mecanismos independientes de la tecnología.

La especificación DPWS define una arquitectura que distingue dos tipos de servicios: los dispositivos y servicios alojados. Los dispositivos juegan una parte importante en el descubrimiento y procedimientos de intercambio de metadatos. Los servicios alojados son servicios web específicos de la aplicación que proporcionan el comportamiento funcional del dispositivo. Estos servicios confían en el dispositivo para cubrir la característica del descubrimiento. El despliegue de los servicios alojados en un dispositivo DPWS es el principal mecanismo de extensibilidad proporcionada por la especificación.

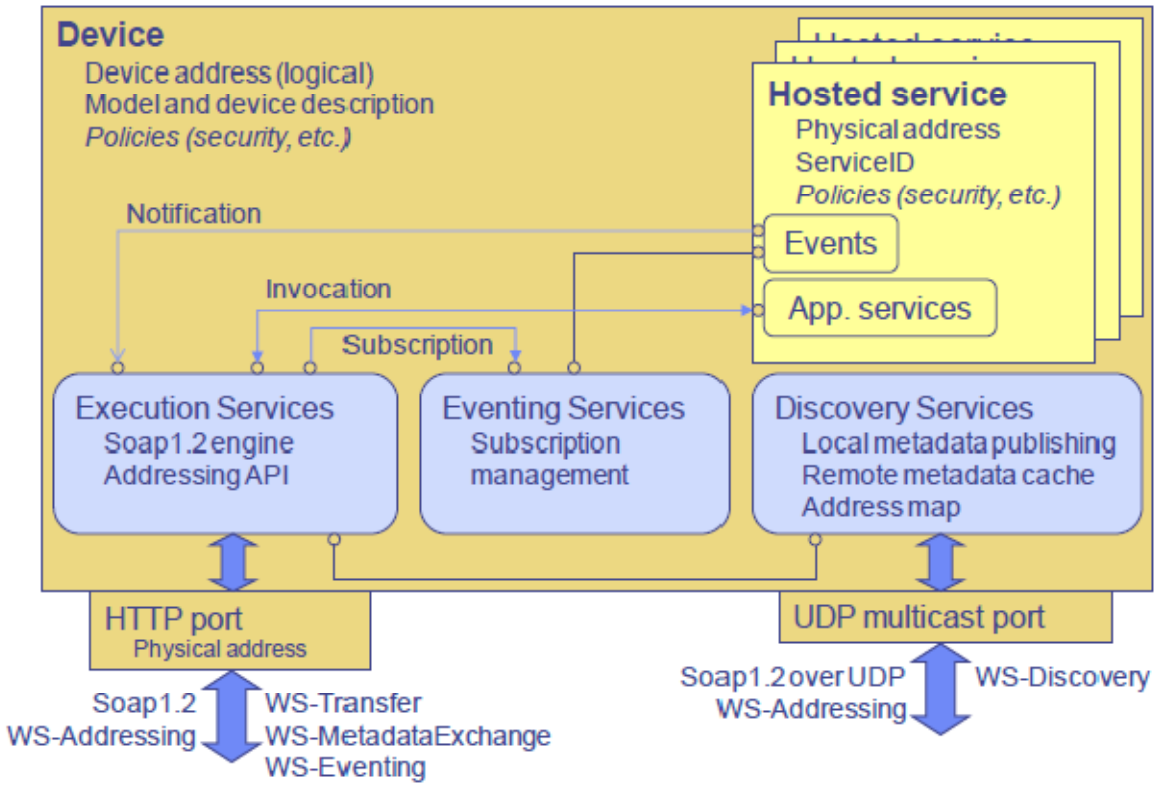


Figura 4.3 Arquitectura DPWS

Todo dispositivo ofrece los siguientes servicios integrados:

- Servicios de descubrimiento (WS-Discovery)
- Servicios de intercambio de metadatos (WS-MetadataExchange)
- Servicios de publicación y suscripción a eventos (WS-Eventing)

Estos servicios sirven de soporte a los servicios propios de la aplicación.

4.3 FRAMEWORKS PARA DESARROLLO DPWS

Para desarrollar software DPWS se han estudiado tres frameworks:

- **WS4D:** WS4D-gSOAP es una extensión del conocido toolkit de Web Services gSOAP, un toolkit para construir Servicios Web basados en SOAP desarrollado en C/C++ por Robert A. van Engelen. Está diseñado para desarrollar servicios web de pequeño tamaño y alto rendimiento.
- **DPWSCore:** Es un proyecto de la iniciativa SOA4D. SOA4D Forge es el sitio de soporte para los desarrolladores y usuarios de las tecnologías SOA4D. SOA4D (Arquitectura Orientada a Servicios para Dispositivos) es una iniciativa de código abierto con el objetivo de fomentar un ecosistema para el desarrollo de componentes de software orientados a servicios (mensajería SOAP y protocolos WS-*, orquestación de servicios...) adaptado a las limitaciones específicas de dispositivos integrados.
- **DPWS Library:** Es un conjunto de librerías del .NET Framework de Microsoft. Este Framework está enfocado al desarrollo para dispositivos de recursos limitados. Las librerías MFWsStack.dll and MFDpwsDevice.dll proporcionan la funcionalidad de alojamiento de servicios para un dispositivo. La librería MFDpwsClient.dll proporciona la funcionalidad para invocar servicios web en dispositivos.

4.4 PROCESO DE DESARROLLO

El proceso general para desarrollar con DPWS consta de los siguientes apartados:

- Definición de los metadatos del dispositivo y del servicio
- Implementación del dispositivo
- Ejecución del dispositivo
- Implementación del cliente
- Ejecución del cliente

Se pueden dar cuatro escenarios de desarrollo:

- **Servicio y dispositivo:** Solo se desarrollan servicios alojados en un dispositivo.
- **Cliente:** Solo se desarrolla el cliente.
- **Peer-to-Peer:** Se desarrolla un servicio que hace de cliente para otro servicio.
- **Asíncrono y eventos:** Se desarrolla un dispositivo (actuando como un cliente) o un cliente puro que necesita recibir mensajes asíncronos de los proveedores de servicios.

Según el caso habrá parte de servidor en el cliente y/o parte de cliente en el servidor.

4.4.1 DEFINICIÓN DEL DISPOSITIVO

DPWS permite definir la descripción del dispositivo en tiempo de ejecución con funciones de la API WS4D-gSOAP. Aunque es mejor alternativa definir la descripción en tiempo de desarrollo y usar el generador de código WS4D-gSOAP para integrar la descripción en el dispositivo.

En los ejemplos de DPWSCore y WS4D-gSOAP se utilizan ejemplos de dispositivos domoticos, como por ejemplo un *aire acondicionado* o la *luz de la cocina*. Aquí se muestra la definición del dispositivo para el *aire acondicionado*:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<wsm:Metadata xmlns:wsm="http://schemas.xmlsoap.org/ws/2004/09/mex"
       xmlns:acs1="http://www.ws4d.org/axis2/tutorial/AirConditioner"
       xmlns:aces1="http://www.ws4d.org/axis2/tutorial/AirConditionerEvent"
       xmlns:wdp="http://schemas.xmlsoap.org/ws/2006/02/devprof"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/discovery"
       xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsm:MetadataSection Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/Relationship">
  <wdp:Relationship Type="http://schemas.xmlsoap.org/ws/2006/02/devprof/host">
   <wdp:Host>
    <wdp:Types>acs1:AirConditioner</wdp:Types>
    <wdp:ServiceId>DPWS-HostingService</wdp:ServiceId>
   </wdp:Host>
   <wdp:Hosted>
    <wdp:Types>acs1:AirConditionerInterface</wdp:Types>
    <wdp:ServiceId>AirConditioner</wdp:ServiceId>
    <Name>AirConditioner</Name>
   </wdp:Hosted>
  </wdp:Relationship>
 </wsm:MetadataSection>
 <wsm:MetadataSection Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisModel">
  <wdp:ThisModel>
   <wdp:Manufacturer lang="de">Universität Rostock</wdp:Manufacturer>
   <wdp:Manufacturer lang="en">University of Rostock</wdp:Manufacturer>
   <wdp:ManufacturerUrl>http://www.uni-rostock.de</wdp:ManufacturerUrl>
   <wdp:ModelName lang="de">Klimaanlagendienst</wdp:ModelName>
   <wdp:ModelName lang="en">AirConditioner Service</wdp:ModelName>
   <wdp:ModelNumber>1.0</wdp:ModelNumber>
   <wdp:ModelUrl>http://www.uni-rostock.de</wdp:ModelUrl>
  </wdp:ThisModel>
 </wsm:MetadataSection>
 <wsm:MetadataSection Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisDevice">
  <wdp:ThisDevice>
   <wdp:FriendlyName lang="de">Klimaanlagendienst</wdp:FriendlyName>
   <wdp:FriendlyName lang="en">AirConditioner Service</wdp:FriendlyName>
   <wdp:FirmwareVersion>Version 0.1</wdp:FirmwareVersion>
   <wdp:SerialNumber>234d5e4f-c23f-4d91-84ba-1f159284b37a</wdp:SerialNumber>
  </wdp:ThisDevice>
 </wsm:MetadataSection>
</wsm:Metadata>
```

La descripción del dispositivo tiene 3 partes: Relationship, ThisModel y ThisDevice. Estos datos son accesibles desde los ordenadores conectados a la misma red. En Windows Vista se pueden ver los dispositivos entrando en Red. En Windows 7 está agrupado en el menú Dispositivos e Impresoras.

Se muestra la detección desde Windows Vista en la siguiente figura:

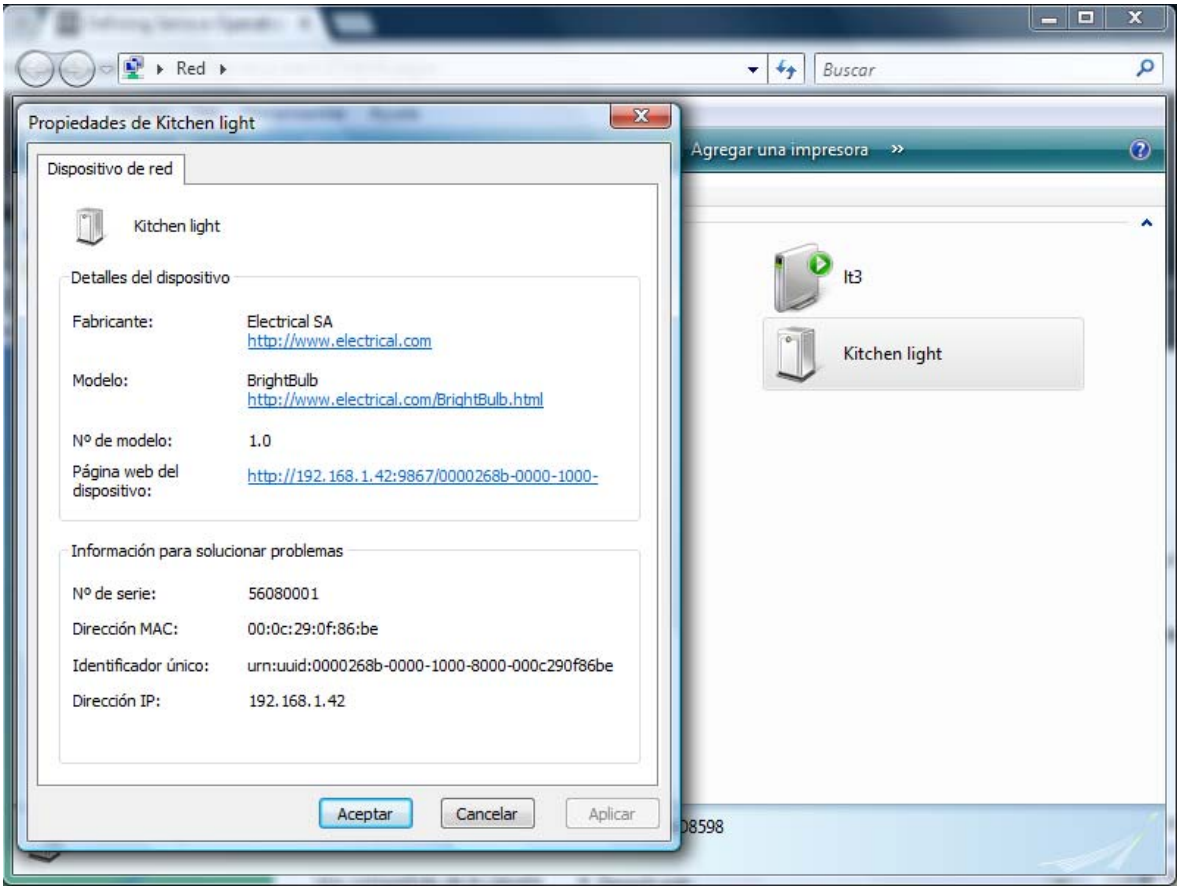


Figura 4.4 Dispositivo descubierto desde Windows Vista

En las propiedades aparecen los detalles del dispositivo: fabricante, modelo, nº de modelo, página web del dispositivo.

NOTA: El dispositivo no es el mismo que el xml de la definición anterior.

El propio dispositivo puede tener una página web:

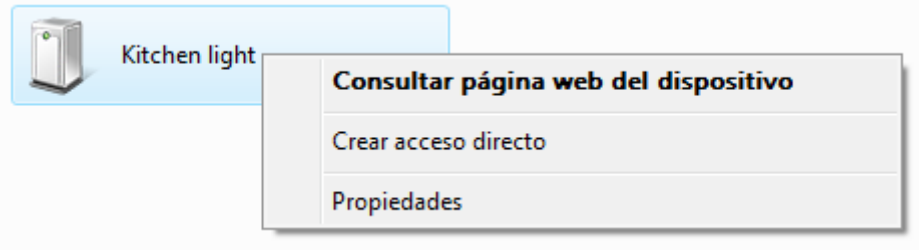


Figura 4.5 Accediendo a la página web del dispositivo

Es muy habitual encontrarse routers con esta funcionalidad. También se puede acceder desde la ventana de propiedades haciendo clic en el vínculo.

Y esta es la web del dispositivo *luz de la cocina*:

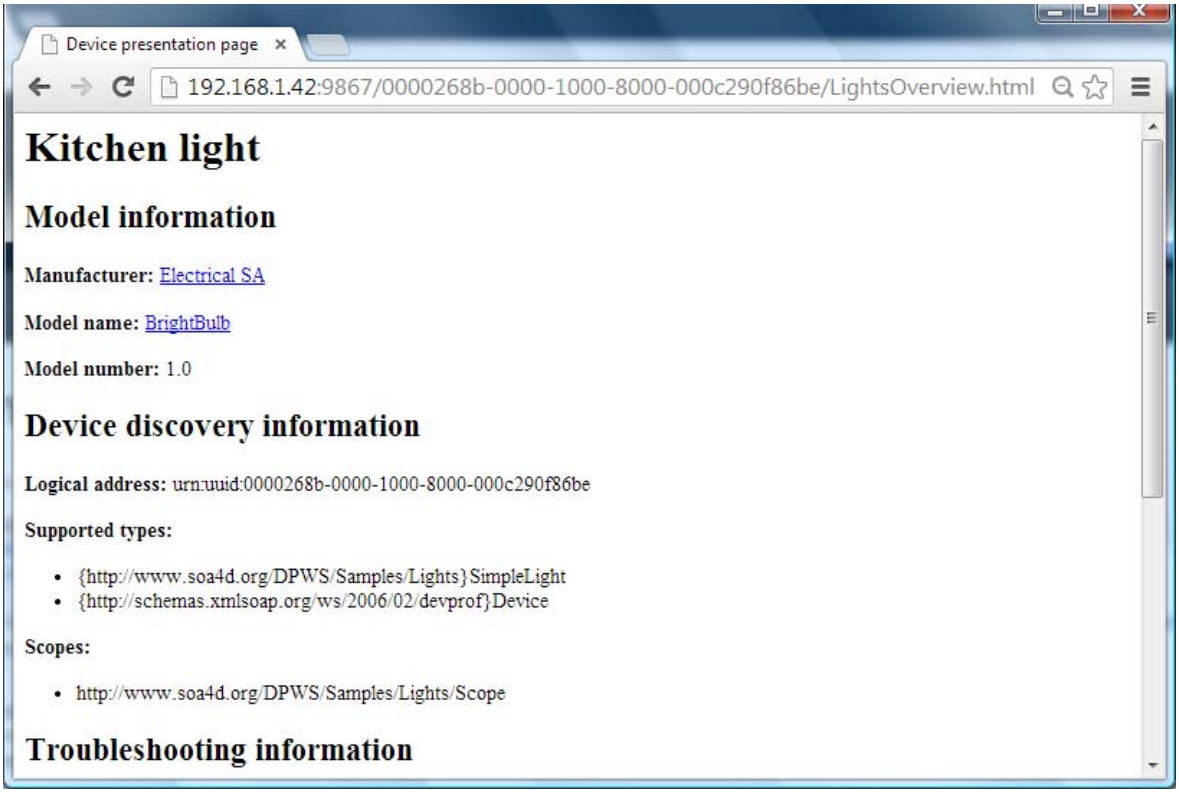


Figura 4.6 Página web del dispositivo

Desde la utilidad DPWS Explorer también se examinan los detalles del dispositivo:

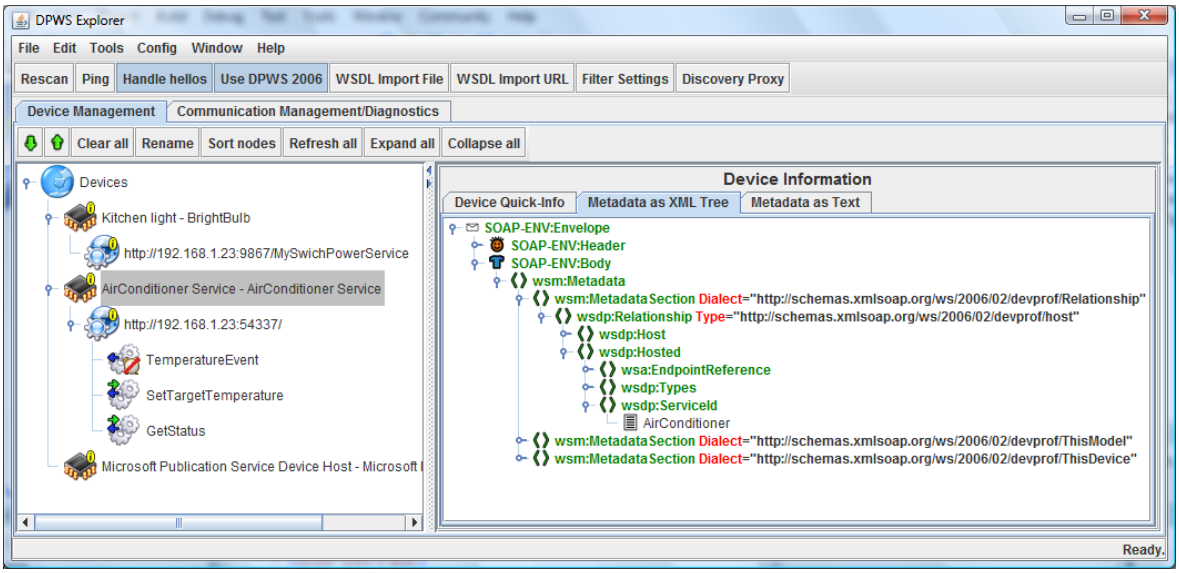


Figura 4.7 Información del dispositivo desde DPWS Explorer

En la figura anterior se ven los dos dispositivos mencionados: Kitchen light y AirConditioner. Cabe destacar que estos dispositivos estaban ejecutándose en una máquina virtual y fueron detectados desde la maquina anfitrión por estar compartiendo la red.

4.4.2 DEFINICIÓN DEL SERVICIO

La manera designada para definir una interfaz de servicio es mediante WSDL.

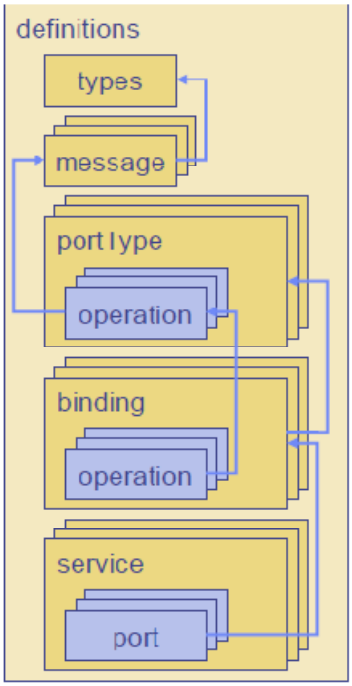


Figura 4.8 Estructura del WSDL

Una herramienta útil para crear archivos WSDL es el componente WSDL Editor del Web Tools Platform (WTP) de Eclipse. A continuación se expone el WSDL del servicio *SwitchPower* del dispositivo *Kitchen Light*:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="Lighting"
       targetNamespace="http://www.soa4d.org/DPWS/Samples/Lights"
       xmlns:tns="http://www.soa4d.org/DPWS/Samples/Lights"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:documentation>
       A simple light control service with one port types:

       1. A simple power ON/OFF port type.
  </wsdl:documentation>

  <!-- TYPES -->

  <wsdl:types>
       <xsd:schema targetNamespace="http://www.soa4d.org/DPWS/Samples/Lights"
                   elementFormDefault="qualified"
                   xmlns:tns="http://www.soa4d.org/DPWS/Samples/Lights">
              <xsd:simpleType name="PowerState">
                     <xsd:restriction base="xsd:token">
                            <xsd:enumeration value="ON" />
                            <xsd:enumeration value="OFF" />
                     </xsd:restriction>
              </xsd:simpleType>

              <xsd:element name="Power" type="tns:PowerState" />
       </xsd:schema>
  </wsdl:types>


  <!-- MESSAGES -->
```

```
  <wsdl:message name="SwitchMsg">
        <wsdl:part name="SwitchBody" element="tns:Power" />
  </wsdl:message>
  <wsdl:message name="GetStatusReqMsg" />
  <wsdl:message name="GetStatusRespMsg">
        <wsdl:part name="StatusBody" element="tns:Power" />
  </wsdl:message>

  <!-- PORT TYPES -->

  <wsdl:portType name="SwitchPower">
        <wsdl:documentation>This port type defines operations for setting the power binary
mode (ON/OFF) of a light device.</wsdl:documentation>
        <wsdl:operation name="Switch">
          <wsdl:documentation>Requests the device to switch off or on its power
state.</wsdl:documentation>
          <wsdl:input message="tns:SwitchMsg" />
        </wsdl:operation>
        <wsdl:operation name="GetStatus">
          <wsdl:documentation>Requests the device to return the value of its power
state.</wsdl:documentation>
          <wsdl:input message="tns:GetStatusReqMsg" />
          <wsdl:output message="tns:GetStatusRespMsg" />
        </wsdl:operation>
  </wsdl:portType>

  <!-- BINDINGS -->

  <wsdl:binding name="SwitchPower" type="tns:SwitchPower">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
          <wsdl:operation name="Switch">
            <wsdl:input>
               <soap:body use="literal" />
            </wsdl:input>
          </wsdl:operation>
          <wsdl:operation name="GetStatus">
            <wsdl:input>
               <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal" />
            </wsdl:output>
          </wsdl:operation>
  </wsdl:binding>

  <!-- Services (Not used by the toolkit) -->

  <wsdl:service name="Lighting">
    <wsdl:port name="SwitchPower" binding="tns:SwitchPower">
       <soap:address location="" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

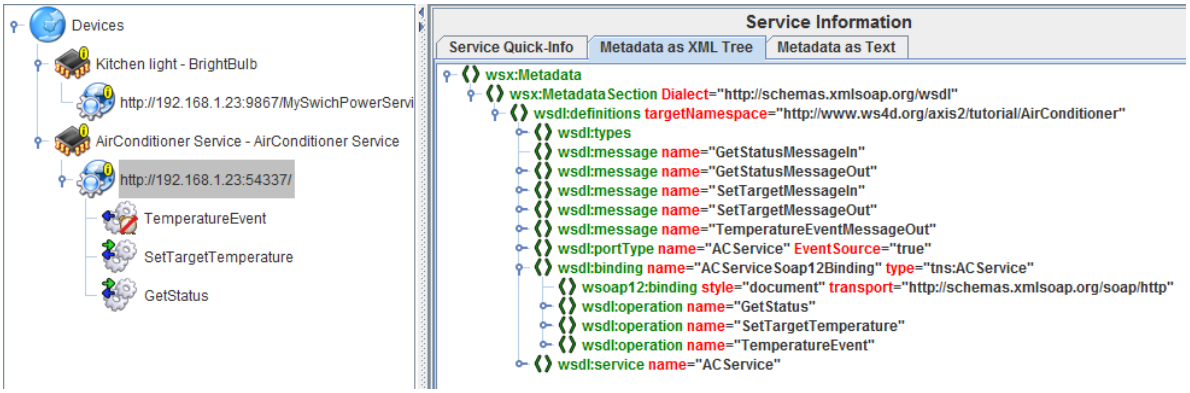


Figura 4.9 Información del servicio desde DPWS Explorer

4.4.3 IMPLEMENTACIÓN DEL DISPOSITIVO

Una vez definido el WSDL se utilizan herramientas de generación de código que nos preparan un skeleton para introducir la lógica del servidor y un stub para introducir la lógica del cliente. WS4D y DPWSCore utilizan gSOAP para generar el código a partir del WSDL.

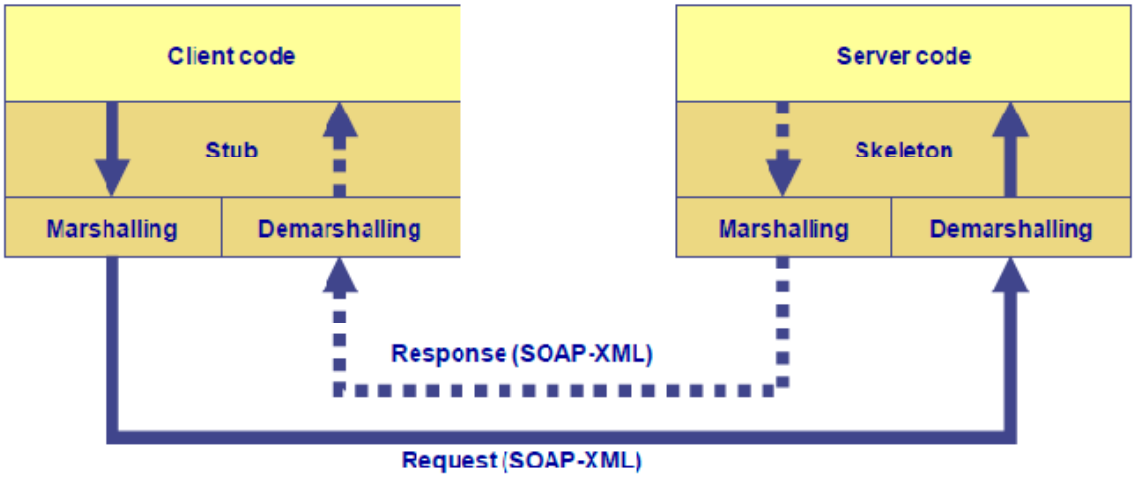


Figura 4.10 Generación de código mediante gSOAP

Para generar el código del aire acondicionado se utiliza este comando:

```
wsdl2h -c -n acs AirConditioner.wsdl -o acs.gsoap
```

Nota: ver el manual de WS4D-gSOAP para mejores referencias. Hay que hacer operaciones manuales sobre el fichero generado antes de ejecutar el siguiente comando.

Este comando genera un fichero acs.gsoap que se utiliza en el siguiente comando:

```
soapcpp2 –2ucn –pws –d gen acs.gsoap
```

Una vez ejecutados estos comandos queda generado el código para rellenar con la lógica del servicio. Tanto en el manual de WS4D como en el de DPWSCore realizan este proceso al detalle. Es recomendable leer los dos porque se complementan muy bien.

4.4.4 EJECUCIÓN DEL DISPOSITIVO

Al ejecutar el código, como es un servidor, se queda escuchando en un puerto TCP. Como se ha visto en la figura 4.3 también escucha un puerto UDP a la espera de mensajes de multidifusión.

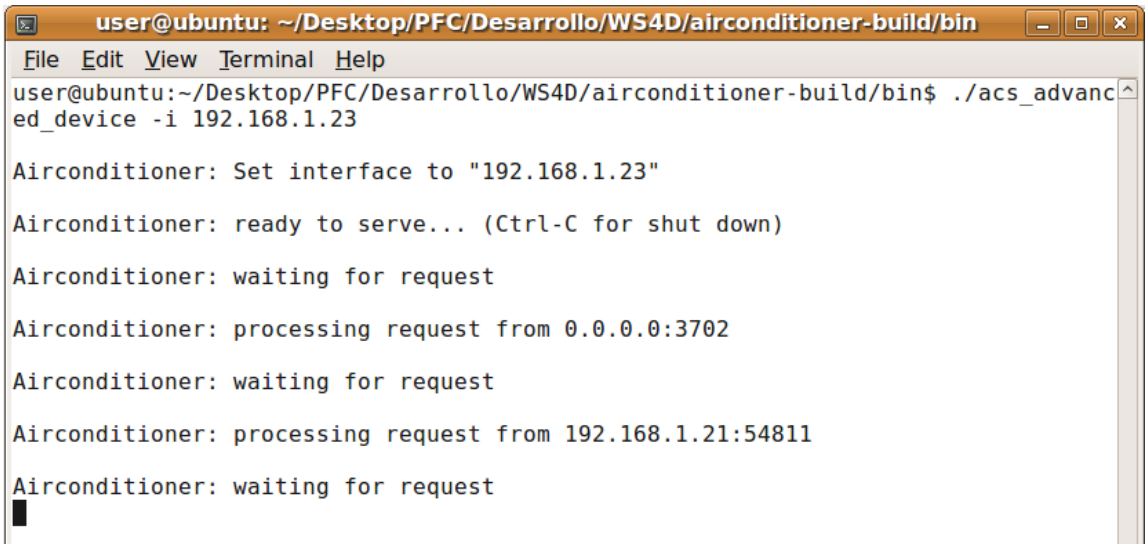


Figura 4.11 Servidor a la espera de peticiones

En DPWS Explorer se pueden invocar acciones. En la siguiente figura se manda un comando para ajustar la temperatura a 22 grados en el *aire acondicionado*:

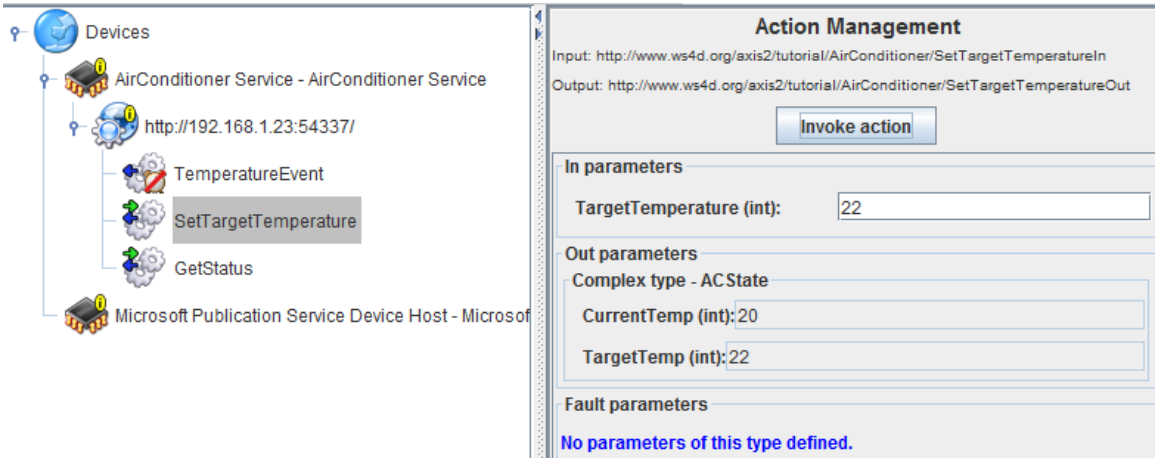


Figura 4.12 Operación desde DPWS Explorer

4.4.5 MONITORIZACIÓN SOAP

Desde DPWS Explorer se puede realizar una monitorización de los mensajes SOAP intercambiados con los dispositivos. Se ven los mensajes de descubrimiento, solicitudes de operaciones, respuesta de operaciones, suscripción a eventos, eventos, etc…

#	Time	Remote IP	Direction	Type	Event
1086	[08:38:07.000321]	239.255.255.250:37...	Outgoing	Multicast	http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe
1087	[08:38:07.000321]	192.168.1.102	Incoming	Request	http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe
1088	[08:38:08.000338]	192.168.1.23	Incoming	Request	http://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches
1089	[08:38:08.000338]	192.168.1.23:45616	Outgoing	Request	http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
1090	[08:38:08.000416]	192.168.1.23:45616	Incoming	Response	http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
1091	[08:38:08.000416]	192.168.1.23:45616	Outgoing	Request	http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
1092	[08:38:08.000416]	192.168.1.23:45616	Incoming	Response	http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
1093	[08:38:08.000837]	192.168.1.21	Incoming	Request	http://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches
1094	[08:38:08.000837]	192.168.1.21:5357	Outgoing	Request	http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
1095	[08:38:08.000853]	192.168.1.21:5357	Incoming	Response	http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
1096	[08:38:15.000723]	192.168.1.23:45616	Outgoing	Request	http://www.ws4d.org/axis2/tutorial/AirConditioner/SetTargetTemperatureIn
1097	[08:38:15.000728]	192.168.1.23:45616	Incoming	Response	http://www.ws4d.org/axis2/tutorial/AirConditioner/SetTargetTemperatureOut
1098	[08:38:18.000878]	192.168.1.23:45616	Outgoing	Request	http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
1099	[08:38:18.000883]	192.168.1.23:45616	Incoming	Response	http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
1100	[08:38:18.000888]	NA	NA	Info	Subscription with id urn:uuid:ab66a750-333f-4506-8e83-4fd66e516401
1101	[08:38:18.000978]	192.168.1.23:45616	Outgoing	Request	http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus
1102	[08:38:18.000983]	192.168.1.23:45616	Incoming	Response	http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse
1103	[08:38:20.000143]	192.168.1.23:37976	Incoming	Request	http://www.ws4d.org/axis2/tutorial/AirConditioner/TemperatureEventOut
1104	[08:38:20.000148]	NA	Incoming	Info	Received event for subscription urn:uuid:ab66a750-333f-4506-8e83-4fd66e5
1105	[08:38:24.000163]	192.168.1.23:45616	Outgoing	Request	http://www.ws4d.org/axis2/tutorial/AirConditioner/GetStatusIn
1106	[08:38:24.000163]	192.168.1.23:45616	Incoming	Response	http://www.ws4d.org/axis2/tutorial/AirConditioner/GetStatusOut
1107	[08:38:25.000153]	192.168.1.23:37977	Incoming	Request	http://www.ws4d.org/axis2/tutorial/AirConditioner/TemperatureEventOut
1108	[08:38:25.000153]	NA	Incoming	Info	Received event for subscription urn:uuid:ab66a750-333f-4506-8e83-4fd66e5
1109	[08:38:30.000163]	192.168.1.23:37978	Incoming	Request	http://www.ws4d.org/axis2/tutorial/AirConditioner/TemperatureEventOut
1110	[08:38:30.000163]	NA	Incoming	Info	Received event for subscription urn:uuid:ab66a750-333f-4506-8e83-4fd66e5
1111	[08:38:34.000683]	192.168.1.23:45616	Outgoing	Request	http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus
1112	[08:38:34.000683]	192.168.1.23:45616	Incoming	Response	http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse
1113	[08:38:35.000553]	192.168.1.23:45616	Outgoing	Request	http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe
1114	[08:38:35.000553]	192.168.1.23:45616	Incoming	Response	http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe
1115	[08:38:35.000883]	NA	Incoming	Info	Subscription end of urn:uuid:ab66a750-333f-4506-8e83-4fd66e516401

Figura 4.13 Monitorización SOAP desde DPWS Explorer

A continuación se visualiza el detalle del evento previo a alcanzar la temperatura solicitada de 22 grados. Cabe notar que la dirección es Incoming y es de tipo Request:

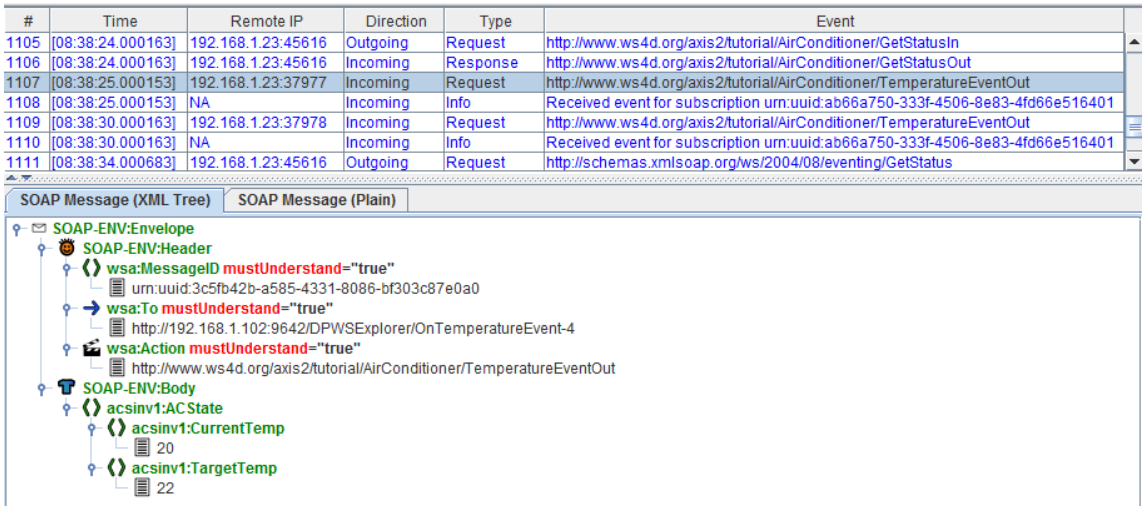


Figura 4.14 Detalle del mensaje SOAP del evento

4.5 PROBLEMAS ENCONTRADOS

Con el ejemplo de la luz de cocina, Kitchen Light, de WS4D el problema encontrado es que no expone las operaciones, se puede ver en la figura 4.9 como aparece el servicio MySwitchPowerService pero no tiene operaciones colgando. Este dispositivo si se detecta desde Windows.

Air Conditioner de DPWSCore no es detectado por Windows. Como en DPWS Explorer aparecen los comandos, se ha realizado la compilación cruzada de este ejemplo. Los respectivos clientes de ejemplo funcionan para los dos frameworks.

Para realizar la compilación de Air Conditioner fue necesario bajar de versión el DPWSCore a la 2.2.0. Aun así aparece un error con los tipos “wchar_t”. Tras revisar una comparativa [22] entre las librerías estándar de C Glibc y uClibc se puede observar que la librería estándar que utiliza la cámara solo soporta UTF-8 y opcionalmente ISO-8859-*, es decir, no soporta los caracteres de 16 bit. Algunas referencias recomiendan actualizar el compilador cris-gcc para solucionar el problema pero esto pone en riesgo el resto del proyecto y no se ha hecho.

Se pueden cargar las dos librerías estándar en la cámara pero desconozco el impacto que puede tener en los programas ya compilados en la cámara y se descartó esta opción.

Con el framework de Microsoft, DPWS Library, hay que implementar la capa de abstracción de hardware HAL quedando descartada por complejidad.

Por estos motivos no se ha podido continuar con la implementación DPWS en la cámara. De cara a la presentación se va a remplazar por un servicio REST para al menos exponer los datos como un servicio.

5 MODELOS PROPUESTOS

En esta sección se muestran unos casos de uso seleccionados para usar los diferentes tipos de servicios DPWS que se pueden ofrecer desde una cámara IP. Primero se repasan las limitaciones de este tipo de sistemas.

5.1 DIFICULTADES TÉCNICAS

En los anteriores capítulos se han visto muchas limitaciones para implementar aplicaciones de visión artificial sobre las cámaras IP. Fundamentalmente han sido limitaciones del hardware de la cámara y limitaciones de los algoritmos de procesado.

Las limitaciones en las cámaras IP han sido principalmente limitaciones de espacio para el software y capacidad de proceso. Se descartó la posibilidad de usarlas en control de calidad en la industria cuando se exigen velocidades extremas. Hemos visto que se pueden implementar aplicaciones de baja velocidad igualmente valiosas.

En las pruebas realizadas durante la realización del presente Proyecto se han conseguido velocidades de 150ms por imagen que es bastante elevada, son 900MB/150ms = 6MB/s realizando un procesado básico. Se estimar una imagen por minuto para procesos más complejos. Se verá un caso de uso en el que la cámara solo realiza un pre-procesado de las imágenes y las envía a un servidor para ser analizadas en profundidad. No es necesario construir todo el sistema en la propia cámara.

5.2 DETECTOR DE MOVIMIENTO

Se usa el modelo implementado en Matlab para detectar movimiento y publicar el evento mediante DPWS. También se mencionará la funcionalidad de detección de movimiento ya incluida en la cámara seleccionada para las pruebas.

El algoritmo de detección de movimiento resulta fácil de implementar en Matlab. Trabaja con una secuencia de imágenes y, en su forma simple, consiste en detectar las diferencias de una imagen con la anterior. La operación es muy sencilla:

```
ImagenDeCambios = abs(ImagenAnterior – ImagenActual)
```

Se trabaja en escala de grises, es decir la luminosidad de cada pixel es comparada con la luminosidad del pixel homólogo en la imagen anterior. Si resultan iguales, consideramos que no hay movimiento y su diferencia es 0. Si algún objeto se mueve introduce una variación en los píxeles que invade y otra en los que abandona. Se suele emplear un margen de error para asegurarse que la variación es suficiente. Esta operación lógica `ImagenDeCambios>25` realiza el proceso de valor umbral (**Thresholding**) que se expuso en el capítulo 3.

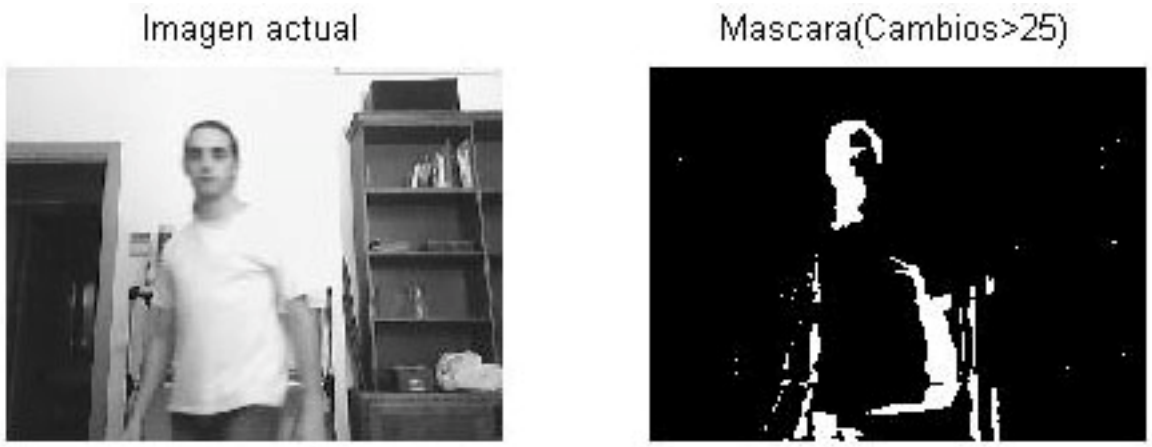


Figura 5.1 Mascara de Cambios en la detección de movimiento

En la anterior imagen aparezco moviéndome hacia la izquierda y desde luego no es la imagen esperada. Se puede pensar que debería aparecer toda la silueta completa marcada como un cambio. Al ser la camiseta de un único color la única diferencia que existe está en los cambios de luminosidad y aunque esté en movimiento no supera el límite establecido, por eso todo el tronco aparece en negro.

Una forma más compleja de realizar el proceso consiste en tomar una imagen del fondo y comparar con la imagen actual, esto puede resultar incluso peor. Pensemos en la entrada de un supermercado. Se puede tomar una imagen antes de abrir las puertas e ir comparando con el resto de imágenes del día. A media mañana el sol iluminará la escena aumentando en la imagen el contraste y se acabaron los cálculos, todo ha variado. Si no es una cosa, es otra, directamente no se puede considerar ninguna imagen como “el fondo”. Lo que se suele hacer es calcular el fondo. Se va a considerar que algo que se quede quieto durante cierto tiempo pasa a formar parte del fondo. Por ejemplo, durante el día se monta un stand de publicidad.

Para el cálculo se realiza una media ponderada entre la imagen actual y el fondo anterior. La imagen actual se va a considerar como un porcentaje del fondo. El porcentaje elegido dependerá de cómo de rápido se requiere que algo estático forme parte del fondo.

```
slowBackground = current * 0.05 + 0.95 * slowBackground;
fastBackground = current * 0.85 + 0.15 * fastBackground;
```

El caso lento tiene mucha memoria, el paso de un objeto deja una estela que permanece mucho tiempo como un cambio. En el caso rápido un objeto que se queda quieto enseguida pasa a considerarse el fondo, habrá que encontrar el equilibrio en función de la aplicación. Para este caso el fondo rápido ha dado buenos resultados.

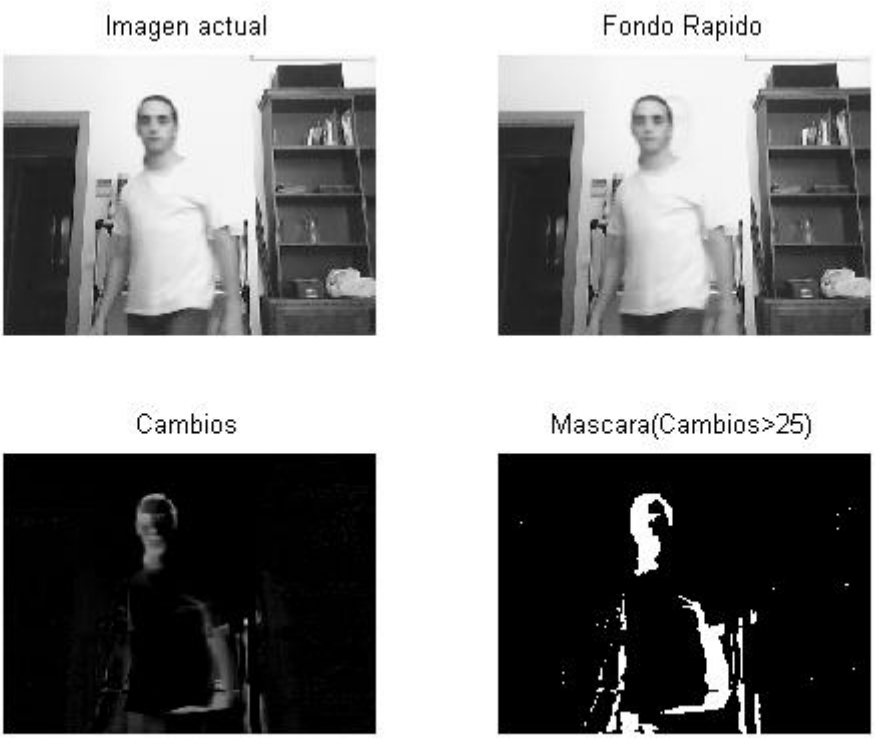


Figura 5.2 Detección de movimiento con Matlab (Resultado)

Para las pruebas se ha usado la cámara web integrada en el portátil. Para conseguir acceso a la imagen hay que realizar una inicialización que se muestra en el código completo que viene a continuación:

```
% Acceso a un dispositivo de adquisicion de imagenes.
vidobj = videoinput('winvideo', 1, 'RGB24_320X240');

% Configurar el dispositivo en el modo de disparo manual.
triggerconfig(vidobj, 'manual');

% Iniciar la adquisicion de imagen.
start(vidobj);

% Esperamos que se caliente.
pause(1);

% Inicializamos el fondo maestro.
current = rgb2gray(getsnapshot(vidobj));
slowBackground = current;
fastBackground = current;
lastBackground = current;
```

Figura 5.3 Detección de movimiento con Matlab (Código 1/2)

La primera parte inicializa las variables antes de entrar en el bucle principal, la segunda parte, se repite durante 5 segundos mostrando la figura con los resultados de cada imagen.

```
tic;
% Durante 5 segundos.
while (toc < 5),
    last = current;
    lastBackground = fastBackground;

    current = rgb2gray(getsnapshot(vidobj));
    % Calculamos el fondo con la media ponderada.
    slowBackground = current * 0.05 + 0.95 * slowBackground;
    fastBackground = current * 0.85 + 0.15 * fastBackground;

    % Calcular la diferencia y mostrar.
    changes = lastBackground - fastBackground;
    moveMask = changes>25;

    %imshow(moveMask);
    figure
    subplot(2,2,1); imshow(current); title('Imagen actual')
    subplot(2,2,2); imshow(fastBackground); title('Fondo Rapido')
    subplot(2,2,3); imshow(changes); title('Cambios')
    subplot(2,2,4); imshow(moveMask); title('Mascara(Cambios>25)')

end
%% parar la camara
stop(vidobj);
```

Figura 5.4 Detección de movimiento con Matlab (Código 2/2)

En la imagen del resultado se pueden apreciar algunos pixeles sueltos que superan el umbral, estas variaciones son producidas por el ruido de la imagen.

En escenas a la intemperie, además del ruido de imagen, se presentan otros inconvenientes: el viento mueve las ramas de los árboles, pueden aparecer bolsas, papeles y pájaros volando, además de otros animales sueltos. Esto requiere introducir un nuevo umbral en la detección, el volumen de los objetos, el número de pixeles agrupados que se encuentran en movimiento. Para evitar a los árboles se introducen las ventanas de inclusión y exclusión de la detección del movimiento, cosa que resulta muy útil en entornos de oficina para iniciar la grabación en el momento que un objeto entra o sale de una de las ventanas definidas. La cámara Axis utilizada en este proyecto trae de serie un algoritmo de detección del movimiento que incluye la complejidad de las ventanas, el tamaño de los objetos y la persistencia de los objetos. Por eso no se va a estudiar más en profundidad este algoritmo en Matlab.

Como se ha descrito al principio de este apartado la detección del movimiento es fácil pero a medida que se avanza en el estudio se descubren numerosas peculiaridades. Es habitual

que aparezcan muchas peculiaridades en las aplicaciones de visión por computador. Hay procesos que para nosotros son extremadamente simples pero cuando se quieren implementar hay que contar con un elevado riesgo técnico. Puede tener una complejidad escondida que imposibilite la realización de un proyecto con los medios materiales disponibles en la actualidad.

En el capítulo 4 se ha mostrado que hay un tipo de servicio que son los eventos, Web Services Eventing (WS-Eventing). En este caso la aplicación cliente, una aplicación de vigilancia, se suscribe al evento "Detectado movimiento" que proporcionan las cámaras de vigilancia.

5.3 CONTADOR DE PERSONAS

En el apartado 3.2.1.1 se vio la implementación del contador de objetos en Matlab. Y se realizó un avance sobre su posible uso para controlar el aforo en las estancias de un museo. Veamos esa idea en mayor detalle.

Figura 5.5 Instantánea de ejemplo de un museo

En esta imagen se observa una escena típica de un museo desde el punto de vista de una cámara de vigilancia. Uno de los primero pasos es segmentar la imagen realizando la conversión a mapa de bits mediante el valor umbral.

Figura 5.6 Problemas durante la segmentación

En la anterior imagen se pueden ver casi todos los problemas de este método de segmentación, aparecen objetos fragmentados y virutas pequeñas (b.3); también aparecen

objetos pequeños (b.2) y posibles huecos (1). No todo objeto ofrece contraste con respecto al fondo.

También puede ocurrir lo contrario, que dos objetos estén en contacto y sean procesados como el mismo objeto. Dependiendo del grado de precisión requerido puede ser suficiente. Se puede realizar el promedio del número de objetos y con bastante seguridad los fragmentos se compensarán con las uniones. Se puede hacer el promedio entre el recuento ofrecido para diferentes valores umbral. Los procesos de rellenado de huecos, la erosión y el filtrado por tamaño elimina los fragmentos pequeños.

Hay que pensar en el resultado del proceso como la señal de un sensor, de hecho es un sensor de imagen. Los valores leídos están sujetos a ruido y se deben promediar y post-procesar. No es razonable que en la sala de un museo entre un fotograma y el siguiente hayan aparecido 25 personas nuevas y a continuación se esfumen.

Figura 5.7 Ejemplo de resultados del contador de personas

Si el nivel de ruido es suficientemente bajo como para dejar ver el patrón de asistencia subyacente es perfectamente válido.

Como se ha visto antes, no es necesario que la cámara esté procesando continuamente las imágenes capturadas. Se puede implementar un servicio “Contar Personas” para ejecutar el proceso bajo demanda.

En el capítulo 2, la figura 2.11 muestra una alternativa para contar las personas que acceden a un recinto (Cross Line Detector). Se puede llegar a implementar el mismo servicio utilizando los datos registrados por dicho algoritmo. Con la diferencia de que la cámara debe estar enfocando al acceso y puede ser más práctico enfocar a toda la sala, por otro lado si se controlan los accesos no es necesario poner una cámara en todas las salas.

5.4 SISTEMA DE CAPTURA DE CARAS

Existen varios algoritmos para detectar caras que van desde la segmentación por color al reconocimiento de caras mediante redes neuronales. Una vez encontrada la cara se puede identificar a la persona mediante una base de datos de caras y un proceso de clasificación.

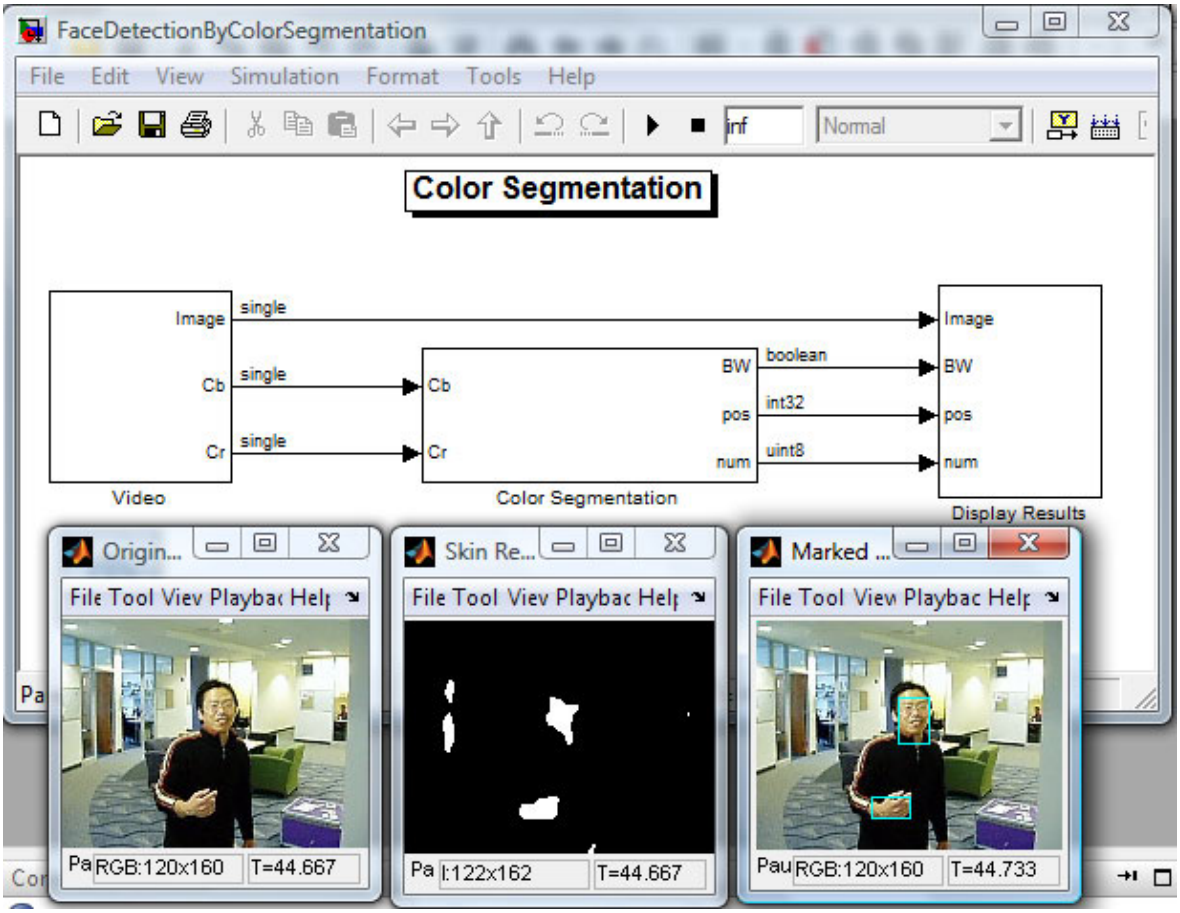


Figura 5.8 Detección de cara por segmentación de color

El problema con la segmentación de color es evidente, las manos son del mismo color, aunque se pueden filtrar por tamaño todavía queda el problema de definir un color para la piel, fácilmente se puede terminar con un algoritmo discriminante por color de piel. No es precisamente el mejor algoritmo pero es aceptable como solución básica.

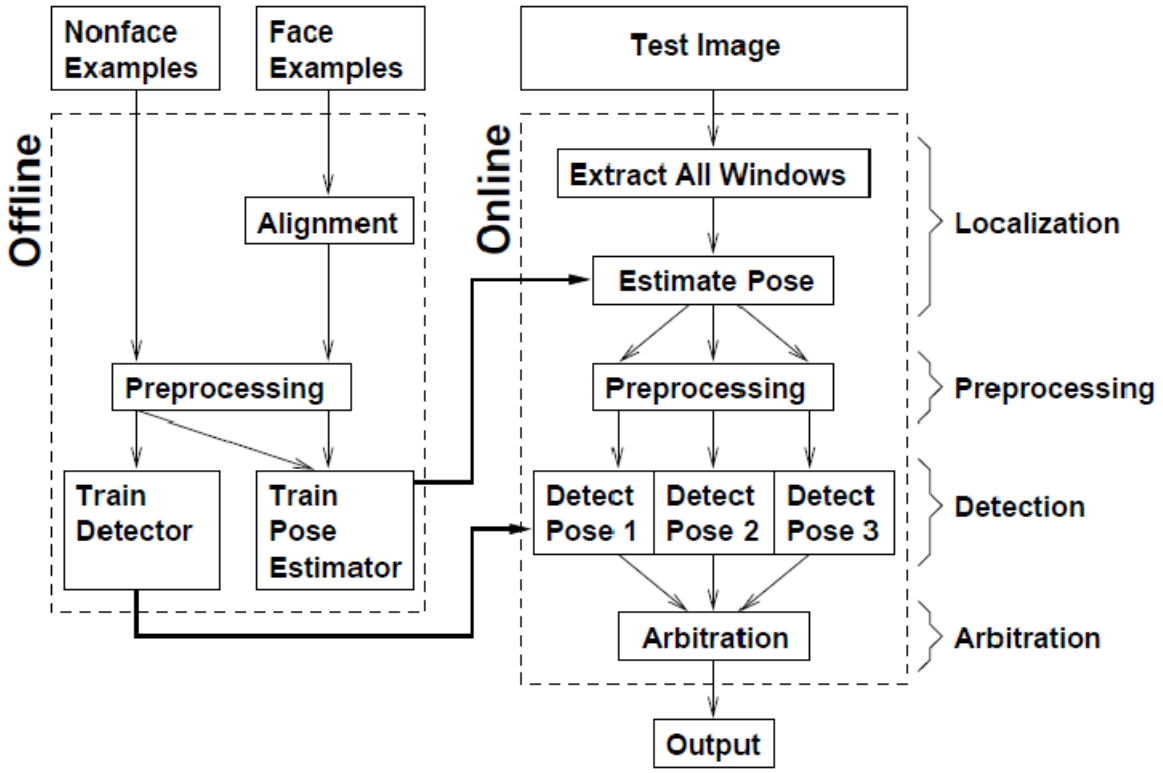


Figura 5.9 Detección mediante redes neuronales

En el diagrama anterior se ven los pasos dados en el sistema de detección de caras basado en redes neuronales implementado por Henry A. Rowley en la referencia [21]. Es un sistema bastante complejo que detecta caras incluso giradas, sin mirar al frente, con contrastes de iluminación, gafas, barba, etc.

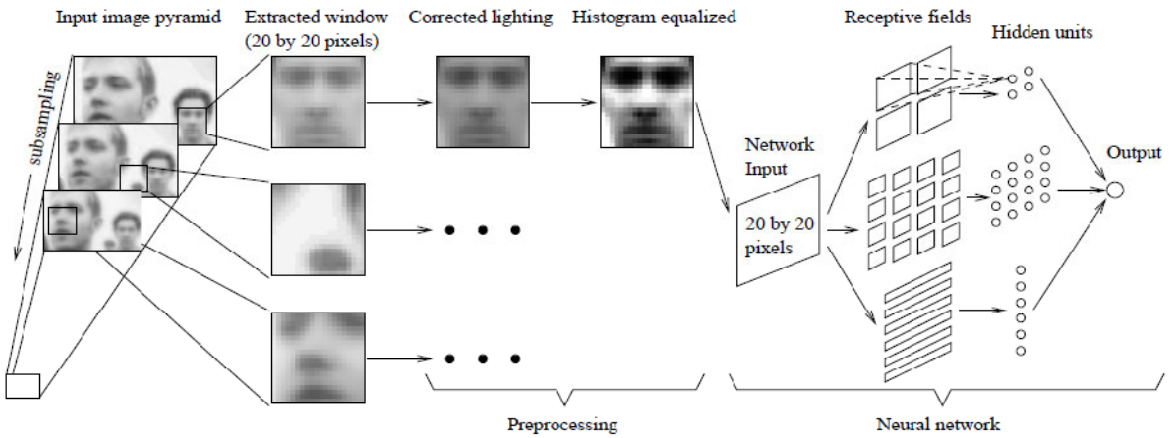


Figura 5.10 Algoritmo básico para detección de caras mediante redes neuronales

Todo proceso de detección tiene 4 casos ampliamente conocidos: positivos, negativos, falsos positivos y falsos negativos. En el caso de la detección de caras salen del cruce entre el valor real y el valor detectado. Si detecta una cara donde no la hay es un falso positivo, si una cara no es detectada es un falso negativo. Para el modelo propuesto no importa tener una mayor tasa de falsos positivos a costa de que no se escape ningún falso negativo. Es preferible tener caras que

no son a que se escapen caras. El cliente realizará un procesado adicional sobre las caras detectadas para buscar coincidencias contra una base de datos.

Este modelo se puede explotar mediante un evento "Cara capturada" a diferencia que el detector de movimiento este servicio envía como adjunto el recorte de la región de la cara como parte de un mensaje MIME. La imagen puede transmitirse sin compresión para mejorar la calidad.

El cliente del servicio debe realizar una detección de patrones sobre las caras recibidas desde la cámara para identificar a los sujetos. Para ello se realiza una correlación cruzada.

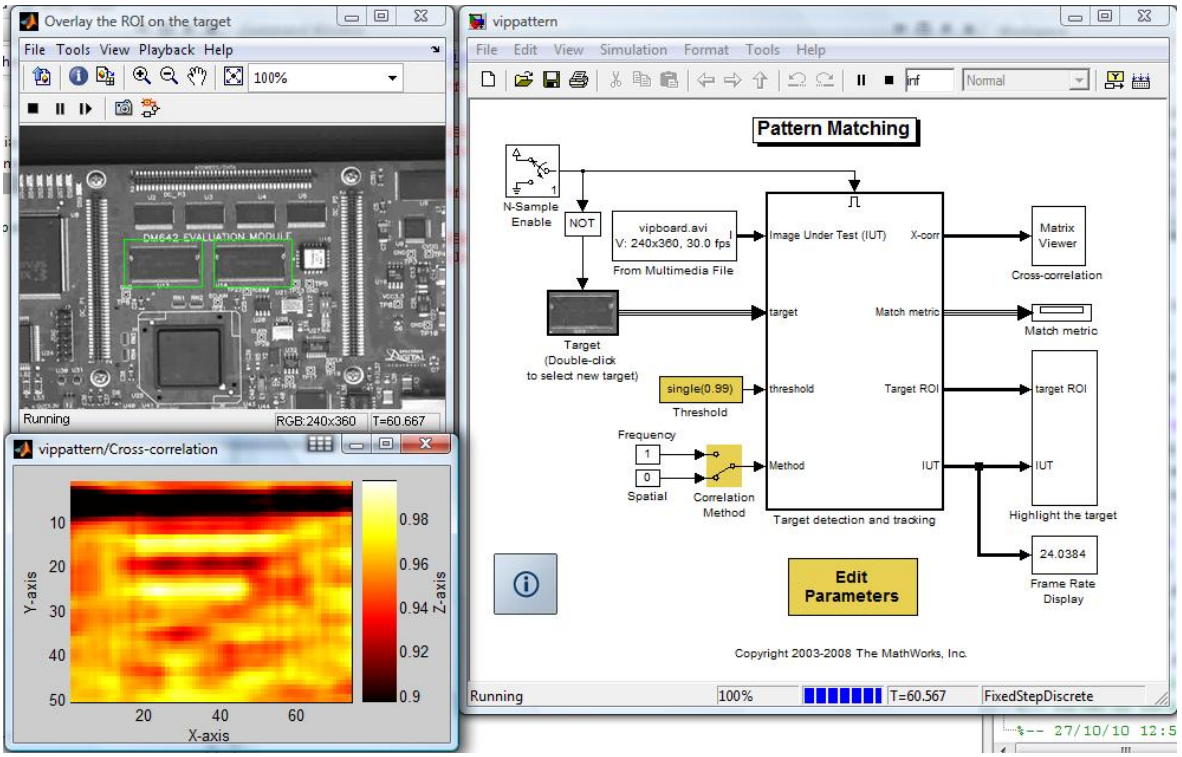


Figura 5.11 Ejemplo de detección de patrones de Simulink

En el ejemplo de detección de patrones de Simulink se realiza la búsqueda de una imagen objetivo (Target) sobre la imagen bajo pruebas (IUT). En la parte inferior izquierda se ve la matriz resultado de la correlación cruzada. Se visualiza con una paleta de colores (hot) que nos permite ver fácilmente las partes "calientes" en "blanco-amarillo" correspondientes a la similitud de esa región con la imagen objetivo. Se fija un umbral de detección (Threshold), en la imagen aparece con el valor 0.99 como entrada al bloque de detección. También se puede seleccionar entre dos métodos de correlación, espacial y en frecuencia.

Como parte de la investigación y para que se vea más claro todo el proceso, rompí el modelo en sus partes básicas quedándome con el método de correlación espacial.

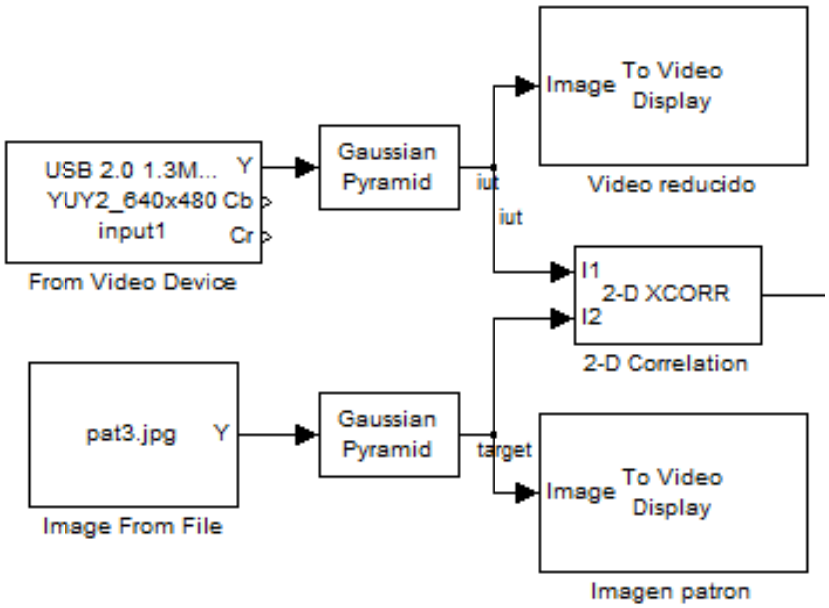


Figura 5.12 Correlación de un patrón contra la imagen de la cámara del portátil (1/2)

En esta primera parte se realiza la correlación cruzada en 2 dimensiones entre la imagen procedente de la cámara usb integrada en el portátil y una imagen patrón, por ejemplo una foto de una cara. En la segunda parte se localizan los máximos locales y se verifica si superan el valor umbral. En ese caso se genera un marco verde sobre la imagen capturada en el lugar donde se han encontrado las coincidencias.

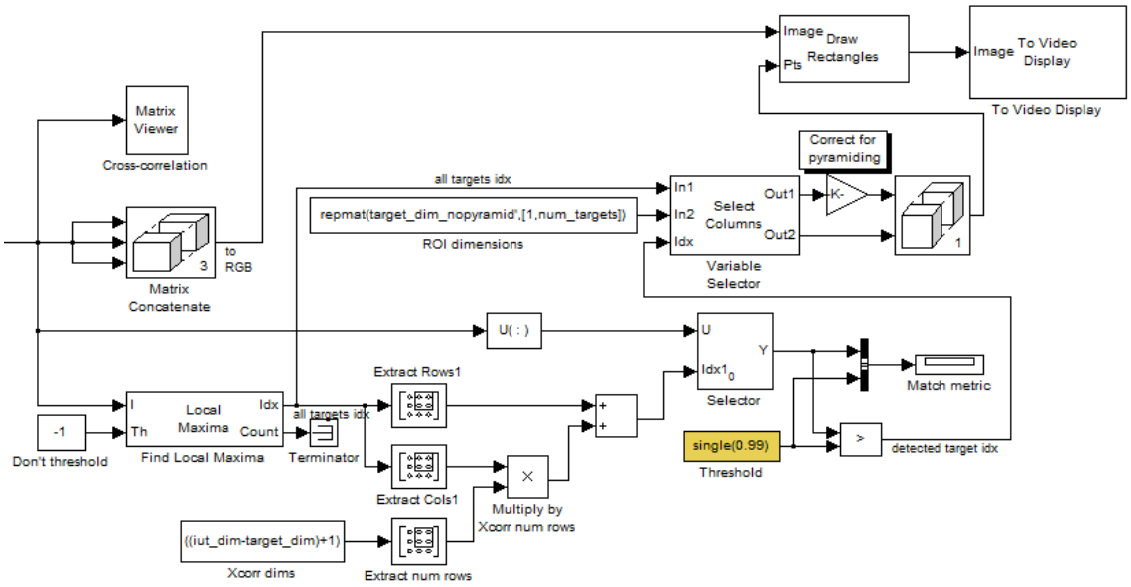


Figura 5.13 Correlación de un patrón contra la imagen de la cámara del portátil (2/2)

5.5 COMPARADOR DEL VOLUMEN DE PIXELES

Es una técnica de fotogrametría diferencial, es decir, en base a una imagen de referencia se mide el número de pixeles que tienen un color seleccionado. Concretamente se mide un valor máximo, uno mínimo y se determina que el número de pixeles entre esas medidas es un valor aceptable. Se puede usar para obtener algunas medidas dimensionales, largo, ancho y en algunos casos concretos el volumen de los objetos. Este es el modelo que se utilizará para definir el caso práctico en el capítulo 6. Consiste en medir el volumen de un líquido de color dentro de una botella transparente como parte de un control de calidad mediante inspección visual.

El proceso consiste en identificar los pixeles con determinado color y para ello se utiliza segmentación por color. Durante las pruebas se obtuvieron resultados no deseados y para mejorar el proceso se ha diseñado un algoritmo específico que cumple muy bien con esta tarea. La descripción del algoritmo se detalla en el punto 3.4.2.1, método ***BMP_FastCountColorProp***.

Para que haya un **color definido** distinto de la gama de grises alguna de las componentes debe destacar sobre las otras. Los cambios de luminosidad modifican las tres componentes por igual. El color pedido (R=70,G=30,B=10) en todo el rango de luminosidad equivale a (R=70+x,G=30+x,B=10+x) donde la x es la alteración de luminosidad entre luces y sombras del objeto. El margen de tolerancia permite ligeros cambios de color al igual que el método anterior. Bajo estos criterios tenemos un sistema de 3 inecuaciones:

```
R - Margen < x + RojoCriterio  < R + Margen
G - Margen < x + VerdeCriterio < G + Margen
B - Margen < x + AzulCriterio  < B + Margen
```

Figura 5.14 Algoritmo de segmentación por color mediante el sistema de inecuaciones

Este criterio funciona muy bien con las luces y sombras presentes en los objetos pero no se puede utilizar para detectar colores en la gama de grises.

```
/* Solucion de un sistema de 3 inecuaciones                    *
 * R - Margen < x + RojoCriterio  < R + Margen                 *
 * G - Margen < x + VerdeCriterio < G + Margen                 *
 * B - Margen < x + AzulCriterio  < B + Margen                 *
 * ColorCriterio pasa restando a ambos lados y                 *
 * se tienen que cumplir todos -margen con todos +margen       *
 * para que el sistema tenga solucion.                         *
 * Eso significa que cumple la proporcion de color             */

if (avgRed   - r - threshold < avgGreen - g + threshold &&
    avgRed   - r - threshold < avgBlue  - b + threshold &&
    avgGreen - g - threshold < avgRed   - r + threshold &&
    avgGreen - g - threshold < avgBlue  - b + threshold &&
    avgBlue  - b - threshold < avgRed   - r + threshold &&
    avgBlue  - b - threshold < avgGreen - g + threshold   )
```

Figura 5.15 Detección del color con sistema de inecuaciones

Para determinar si un color cumple el criterio, el sistema de inecuaciones debe tener solución, es decir, se verifican todas las condiciones 2 a 2. Los avg son promediados 3x3 en la medida de las componentes de color.

El resultado del método ***BMP_FastCountColorProp*** devuelve el número estimado de pixeles que tienen ese color. Este método es llamado desde un componente que agrupa las estadísticas de comparación contra los niveles máximo y mínimo definidos como parte de la configuración. Este componente es el que ofrece el servicio de "Estadísticas de Comparación" arrojando como resultado un pequeño resumen con el número de objetos que superan el máximo, número de objetos por debajo del umbral mínimo y número de objetos que están entre los márgenes.

Al agrupar un servicio de estadísticas se agrega toda la responsabilidad del control de calidad a la cámara. Incluso se puede hacer para que la propia cámara active un relé y descarte los objetos con mediciones fuera de los márgenes.

Conclusiones:

Resulta fácil proponer servicios para implementar en las cámaras. Pueden ser bajo demanda o mediante eventos. Se pueden publicar resultados parciales repartiendo la capacidad de cálculo entre el cliente y la cámara. Se optimiza el consumo de ancho de banda y en algunos casos se puede transmitir solo la información necesaria sin necesidad de compresión y con la consecuente ganancia en calidad.

6 CASO PRÁCTICO

En esta sección se describe al detalle el algoritmo implementado para la demostración.

6.1 USO PRÁCTICO DEL COMPARADOR DE VOLÚMENES

En el capítulo anterior se ha visto un modelo propuesto para medir el volumen de píxeles que ocupa un objeto en la imagen y así establecer una relación entre una medida física. Se habló de su uso para medir el volumen de un líquido con un color definido en un recipiente transparente.

6.1.1 CONTROL DE LLENADO DE RECIPIENTES

En este caso práctico se va a utilizar para controlar el nivel de llenado de una botella de aceite en una aplicación de control de calidad industrial. La imagen a procesar es una botella de aceite sobre una cinta transportadora y un fondo blanco como se muestra en este esquema:

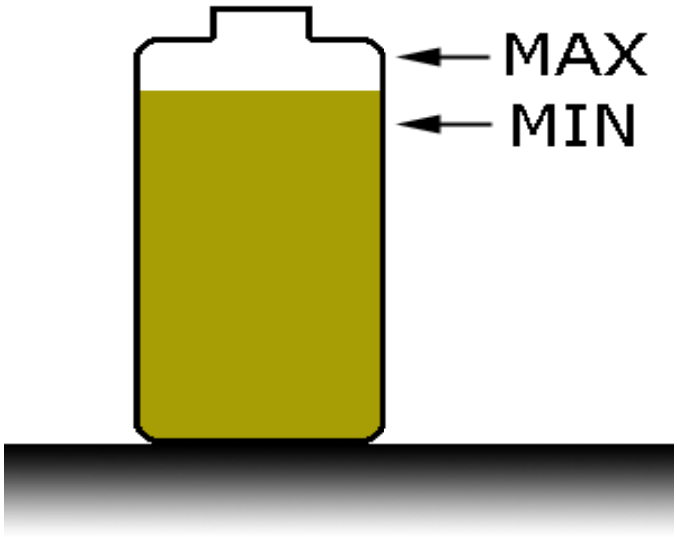


Figura 6.1 Esquema del control del nivel de llenado

Como se puede ver en la imagen el área de color amarillo-verdoso ocupa un número determinado de píxeles que se puede medir. Si la botella tuviera un nivel inferior el área de color se contabiliza con menor número de píxeles y viceversa.

Si la botella estuviera ligeramente desplazada en la cinta no influye en la medida, pero las condiciones de iluminación y la velocidad de la botella pueden afectar negativamente por eso es recomendable utilizar una iluminación artificial y que la botella lleve una velocidad reducida. El proceso de configuración se debe realizar en las mismas condiciones.

6.1.2 MOTIVOS DE LA ELECCIÓN

El detector de movimiento ya viene incorporado en la cámara, tanto el contador de personas como el sistema de captura de caras tienen algoritmos complejos para ser implementados sin la ayuda de librerías de procesado de imágenes. Para evitar complicaciones en la implementación el algoritmo más simple era el comparador de volúmenes.

6.2 MODELO DEL SISTEMA

En esta sección se realiza la descripción completa del sistema. Desde el montaje a la explotación del servicio. Se describe también el software necesario y las comunicaciones.

6.2.1 ESQUEMA DE MONTAJE

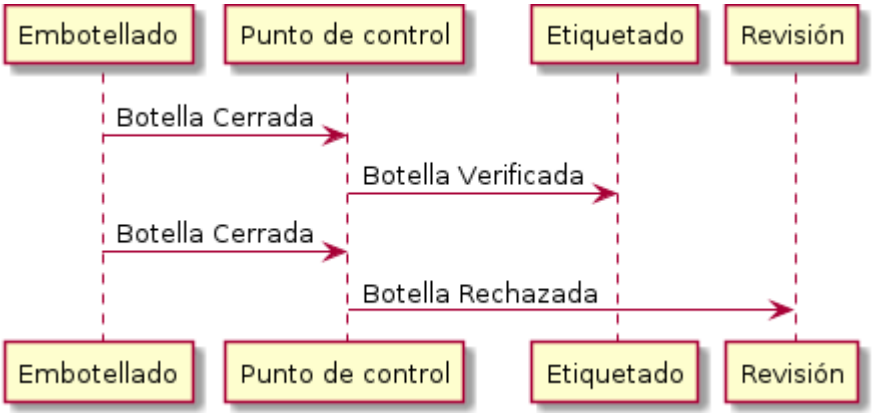


Figura 6.2 Diagrama de secuencia en línea de producción

Para montar este sistema en un entorno productivo hay que seguir estos pasos:

1. Preparar un slot de un metro para insertar el punto de control en la línea de producción tras el rellenado y cierre de las botellas. Colocar una carcasa opaca rodeando la línea de transporte. En su interior colocar una fuente de luz difusa y un fondo blanco en uno de los laterales.
2. En el otro lateral de la carcasa colocar la cámara IP. Suministrarle alimentación y conexión por red Ethernet. Opcionalmente se puede usar el puerto de control de relés de la propia cámara para activar el mecanismo de expulsión de la línea. La imagen de la cámara debe ser similar a la figura 6.1, no importa que se visualice la botella al completo, basta con que se vea la parte superior donde están los niveles de llenado.
3. Las botellas deben atravesar este punto de control de una en una y deben realizarlo a velocidad reducida. Durante el proceso de configuración deben darse las mismas condiciones.

4. En el centro de control se utilizará una computadora como cliente del servicio. Esta computadora debe tener conexión a la red local. La configuración del sistema solo hay que hacerla una vez, se puede realizar desde esta misma computadora.
5. Preparar 2 botellas modelo con los niveles máximo y mínimo. El técnico utilizará la aplicación setup para capturar las imágenes y seleccionar el color.
6. Preparar 3 botellas modelo con un nivel inferior al mínimo, un nivel intermedio y superior al máximo. Las 5 botellas se usarán para hacer una secuencia de verificación.
7. Tras la secuencia de verificación se resetean las estadísticas y se da por concluida la puesta a punto.

6.2.2 DIAGRAMA DE CASOS DE USO

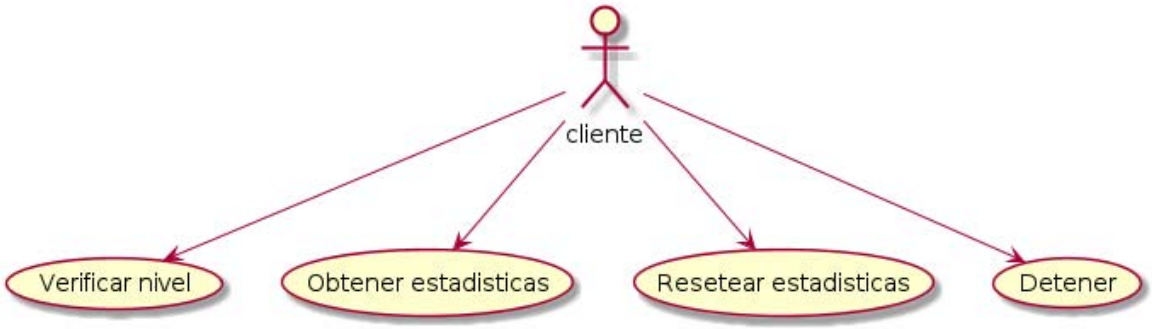


Figura 6.3 Diagrama de casos de uso

La aplicación cliente puede realizar 4 operaciones básicas:

- **Verificar nivel:** Permite realizar medidas independientes.
- **Obtener estadísticas:** Recupera los resultados estadísticos desde el último restablecimiento de estadísticas.
- **Resetear estadísticas:** Reinicia los contadores de resultados estadísticos.
- **Detener:** Detiene el servicio de forma ordenada. Almacena las estadísticas de forma permanente en el sistema de ficheros.

6.2.3 DIAGRAMA DE COMPONENTES

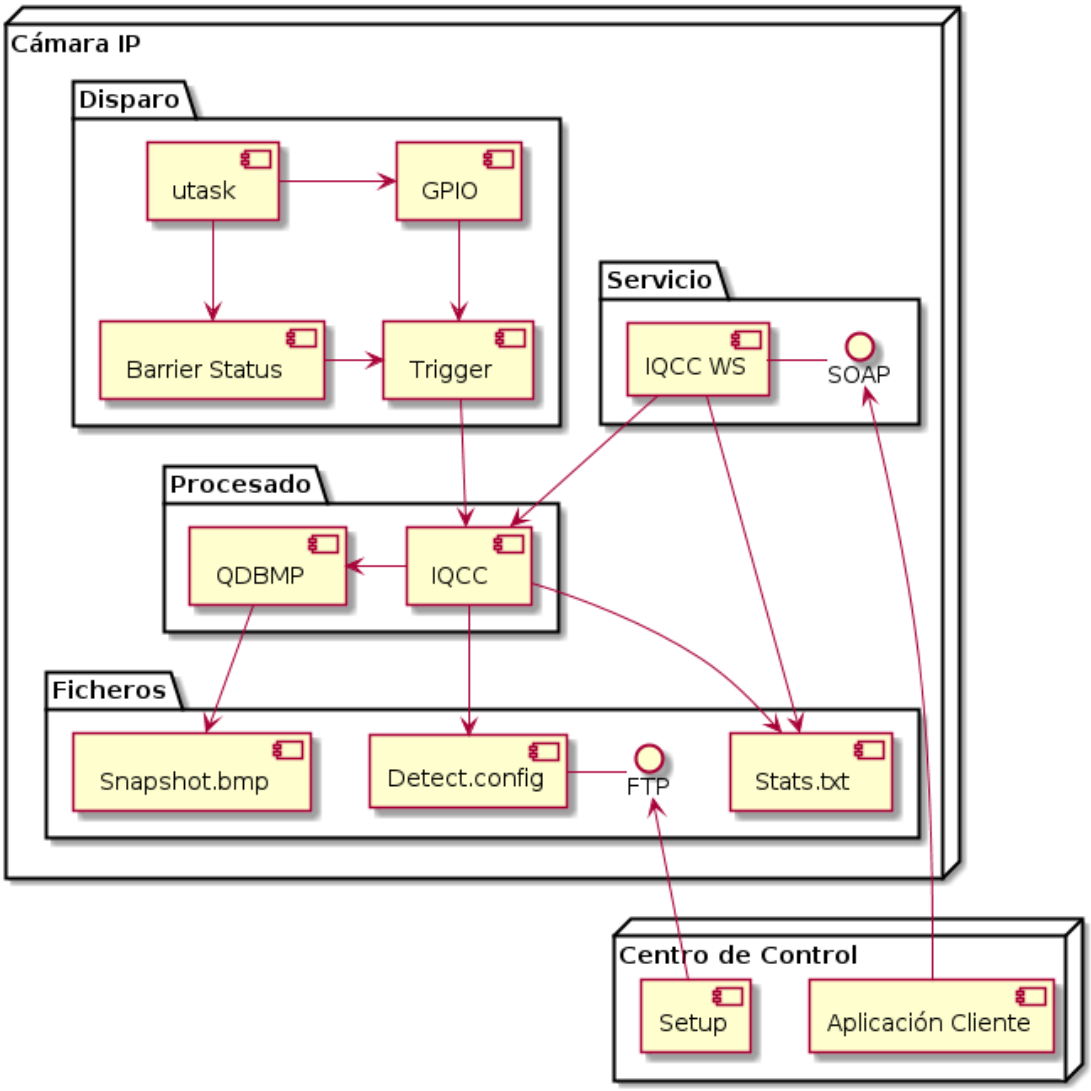


Figura 6.4 Diagrama de componentes

El Centro de Control necesita dos componentes software:

- **Aplicación Setup:** Utilidad que permite al operador visualizar las instantáneas de la cámara y seleccionar sobre ellas los casos máximo y mínimo. Transfiere a la cámara el fichero con la configuración por FTP.
- **Aplicación Cliente:** Aplicación desarrollada por terceros que toma el rol de cliente contra el componente IQCC. La comunicación entre estos componentes se realiza mediante Web Services SOAP y la interfaz del servicio es la descrita por el diagrama de casos de uso anterior.

En la Cámara IP se encuentran 4 paquetes de software:

- **Disparo:** Este paquete de programas es el encargado de ejecutar el procesado de imagen cuando se detecta un cruce de barrera. **GPIO** gestiona la barrera por infrarrojos leyendo el puerto externo de la cámara a la espera de una señal de circuito cerrado. **Barrier Status**, implementado en el script toogle, presenta una interfaz web y por línea de comandos que permite activar o desactivar la detección de movimiento (barrera virtual). **Trigger** componente que centraliza las llamadas producidas por los eventos de cruce de barrera e invoca al componente de calidad IQCC. **utask** componente del sistema Linux que se utiliza para inicializar la configuración de los eventos cuando se reinicia la cámara.
- **Procesado:** Este paquete consta del componente **Image Quality Control Component (IQCC)** que realiza el control estadístico. Utiliza la librería **QDBMP** que es la encargada de realizar el procesamiento de la imagen capturada Snapshot.bmp.
- **Servicio: IQCC WS** es el Web Service de IQCC que proporciona los resultados estadísticos como parte del servicio.

6.2.4 DIAGRAMA DE CLASES

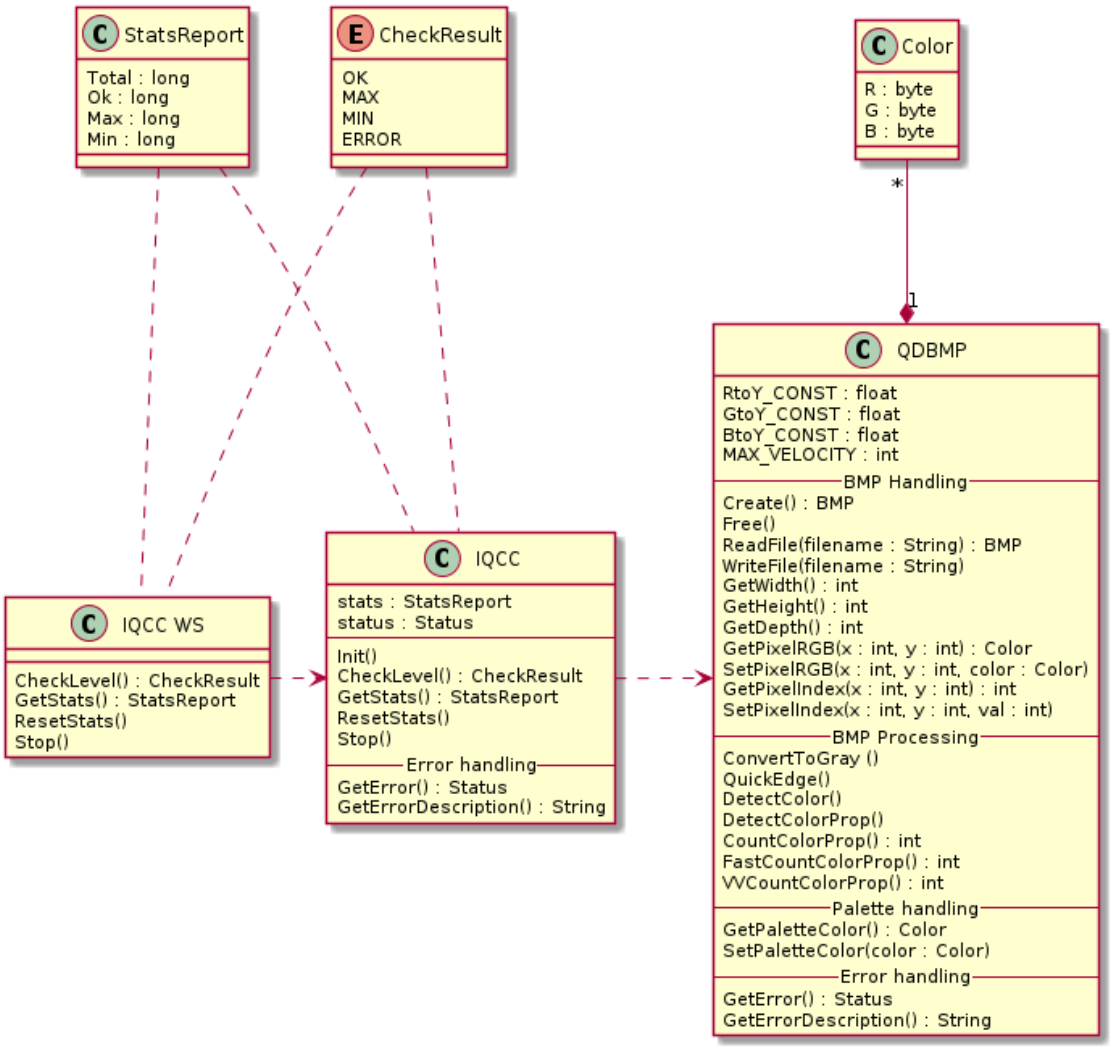


Figura 6.5 Diagrama de clases

Este diagrama es una representación conceptual del paquete de procesado. Al estar implementado en C no resulta trivial su representación en UML. **QDBMP** hace las funciones de clase BMP y las funciones de procesado de la librería como se describió en 3.4.2.1. **IQCC** lleva el control de las estadísticas y se encarga de almacenarlas de forma persistente en disco, es capaz de traducir el número de pixeles a un resultado utilizando la configuración Detect.config enviada por la aplicación setup. **IQCC WS** proporciona el servicio como Web Service SOAP.

Nota: Todos los métodos y propiedades de QDBMP aparecen en el código precedidos por "BMP_" y en IQCC van precedidos por "IQCC_". Se han eliminado del diagrama para mayor claridad.

6.2.5 DIAGRAMAS DE SECUENCIA

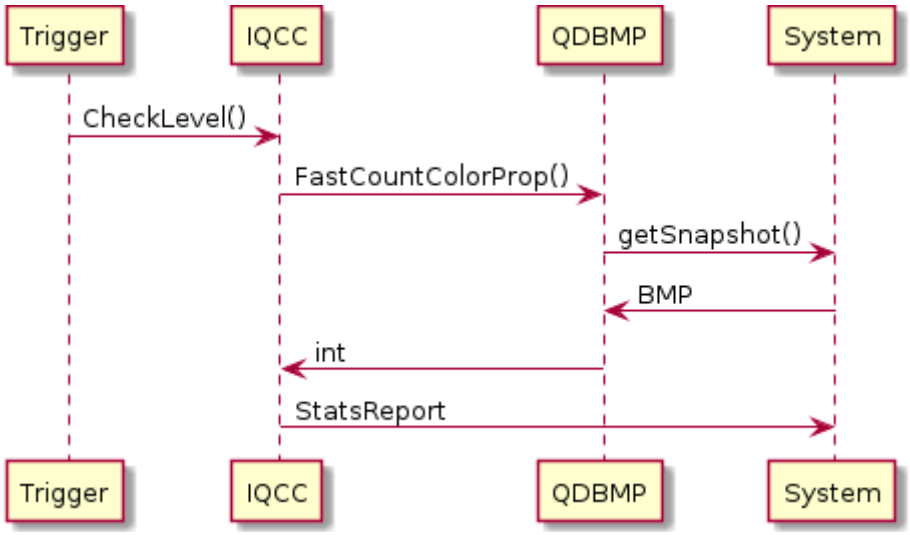


Figura 6.6 Diagrama de secuencia del disparo

En este diagrama se puede ver la interacción entre los diferentes componentes cuando se produce un disparo al cruzar la botella una de las barreras. System es la plataforma Linux provista en la cámara, getSnapshot() en la práctica se realiza invocando a un comando proporcionado por el fabricante que convierte la imagen JPEG procedente del sensor en la imagen Snapshot.bmp

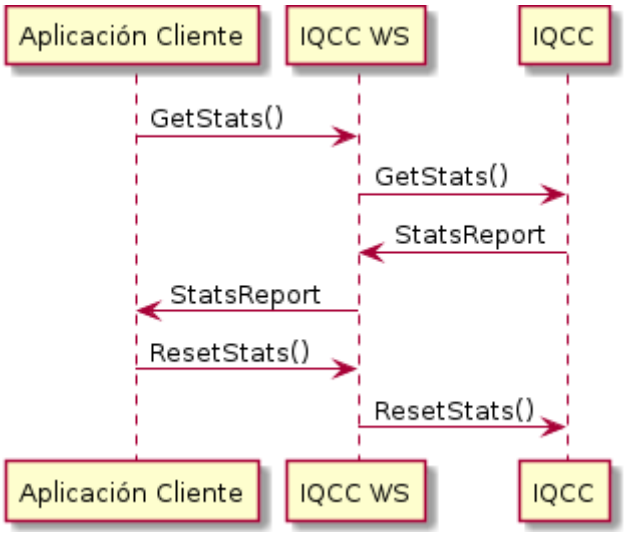


Figura 6.7 Diagrama de secuencia del servicio

Aquí se ve la interacción entre los componentes cuando se invoca el Web Service. La aplicación cliente se comunica mediante SOAP con el componente IQCC WS.

6.2.6 DEFINICIÓN DEL SERVICIO

A continuación se muestra la definición del servicio en WSDL:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
       name="IQCC"
       targetNamespace="http://www.euitt.upm.es/DPWS/IPCam/Quality"
       xmlns:tns="http://www.euitt.upm.es/DPWS/IPCam/Quality"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <wsdl:documentation>
              IQCC - Image Quality Control Component
       </wsdl:documentation>

<!--    TYPES          -->

  <wsdl:types>
    <xsd:schema elementFormDefault="qualified" targetNamespace="http://www.euitt.upm.es/">
      <xsd:element name="IQCC_CheckLevel">
        <xsd:complexType />
      </xsd:element>
      <xsd:element name="IQCC_CheckLevelResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element     minOccurs="1"     maxOccurs="1"     name="IQCC_CheckLevelResult"
type="tns:IQCC_CHECK_RESULT" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:simpleType name="IQCC_CHECK_RESULT">
        <xsd:restriction base="s:string">
          <xsd:enumeration value="IQCC_CHECK_OK" />
          <xsd:enumeration value="IQCC_CHECK_MAX" />
          <xsd:enumeration value="IQCC_CHECK_MIN" />
          <xsd:enumeration value="IQCC_CHECK_ERROR" />
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:element name="IQCC_GetStats">
        <xsd:complexType />
      </xsd:element>
      <xsd:element name="IQCC_GetStatsResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element     minOccurs="0"     maxOccurs="1"     name="IQCC_GetStatsResult"
type="tns:IQCC_STATS_REPORT" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:complexType name="IQCC_STATS_REPORT">
        <xsd:sequence>
          <xsd:element minOccurs="1" maxOccurs="1" name="Total" type="s:unsignedLong" />
          <xsd:element minOccurs="1" maxOccurs="1" name="Ok" type="s:unsignedLong" />
          <xsd:element minOccurs="1" maxOccurs="1" name="Max" type="s:unsignedLong" />
          <xsd:element minOccurs="1" maxOccurs="1" name="Min" type="s:unsignedLong" />
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="IQCC_ResetStats">
        <xsd:complexType />
      </xsd:element>
      <xsd:element name="IQCC_ResetStatsResponse">
        <xsd:complexType />
      </xsd:element>
    </xsd:schema>
  </wsdl:types>

<!--    MESSAGES       -->

  <wsdl:message name="IQCC_CheckLevelSoapIn">
    <wsdl:part name="parameters" element="tns:IQCC_CheckLevel" />
  </wsdl:message>
  <wsdl:message name="IQCC_CheckLevelSoapOut">
    <wsdl:part name="parameters" element="tns:IQCC_CheckLevelResponse" />
  </wsdl:message>
```

```
  <wsdl:message name="IQCC_GetStatsSoapIn">
    <wsdl:part name="parameters" element="tns:IQCC_GetStats" />
  </wsdl:message>
  <wsdl:message name="IQCC_GetStatsSoapOut">
    <wsdl:part name="parameters" element="tns:IQCC_GetStatsResponse" />
  </wsdl:message>
  <wsdl:message name="IQCC_ResetStatsSoapIn">
    <wsdl:part name="parameters" element="tns:IQCC_ResetStats" />
  </wsdl:message>
  <wsdl:message name="IQCC_ResetStatsSoapOut">
    <wsdl:part name="parameters" element="tns:IQCC_ResetStatsResponse" />
  </wsdl:message>

<!--   PORT TYPES     -->

  <wsdl:portType name="IQCCSoap">
    <wsdl:operation name="IQCC_CheckLevel">
      <wsdl:input message="tns:IQCC_CheckLevelSoapIn" />
      <wsdl:output message="tns:IQCC_CheckLevelSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="IQCC_GetStats">
      <wsdl:input message="tns:IQCC_GetStatsSoapIn" />
      <wsdl:output message="tns:IQCC_GetStatsSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="IQCC_ResetStats">
      <wsdl:input message="tns:IQCC_ResetStatsSoapIn" />
      <wsdl:output message="tns:IQCC_ResetStatsSoapOut" />
    </wsdl:operation>
  </wsdl:portType>

<!--   BINDINGS       -->

  <wsdl:binding name="IQCCSoap" type="tns:IQCCSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="IQCC_CheckLevel">
      <soap:operation soapAction="http://www.euitt.upm.es/IQCC_CheckLevel" style="document"
/>
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="IQCC_GetStats">
      <soap:operation  soapAction="http://www.euitt.upm.es/IQCC_GetStats"  style="document"
/>
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="IQCC_ResetStats">
      <soap:operation soapAction="http://www.euitt.upm.es/IQCC_ResetStats" style="document"
/>
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

<!--   BINDINGS (SOAP 1.2)   -->

  <wsdl:binding name="IQCCSoap12" type="tns:IQCCSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="IQCC_CheckLevel">
      <soap:operation soapAction="http://www.euitt.upm.es/IQCC_CheckLevel" style="document"
/>
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
```

```
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="IQCC_GetStats">
      <soap:operation  soapAction="http://www.euitt.upm.es/IQCC_GetStats"  style="document"
/>
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="IQCC_ResetStats">
      <soap:operation soapAction="http://www.euitt.upm.es/IQCC_ResetStats" style="document"
/>
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

<!--   Services      -->

  <wsdl:service name="IQCC">
    <wsdl:port name="IQCCSoap" binding="tns:IQCCSoap">
      <soap:address location="http://localhost:50019/IQCC.asmx" />
    </wsdl:port>
    <wsdl:port name="IQCCSoap12" binding="tns:IQCCSoap12">
      <soap:address location="http://localhost:50019/IQCC.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

6.3 IMPLEMENTACIÓN DEL SISTEMA

En este apartado se muestran detalles de la instalación y configuración del sistema.

6.3.1 SOFTWARE EN EL CLIENTE

Como se ha expuesto en el diagrama de componentes el centro de control utiliza dos piezas de software, la aplicación cliente del servicio que deben implementar terceras partes para integrarlo con el software de control de la compañía y la aplicación setup que se proporciona con el sistema.

Figura 6.8 Aplicación Setup

En la figura anterior se observa un caso en el que se ha utilizado agua con una ligera coloración y se ha iluminado deficientemente a propósito para ver la eficacia del algoritmo. En la parte inferior aparece la sobra de la botella y es capaz de distinguirla cuando está fuera de la botella. La parte de sombra que se ve a través de la botella y no perjudica la detección.

6.3.2 SOFTWARE EN LA CÁMARA IP

La instalación de software en la cámara IP se puede realizar fácilmente restaurando un backup en una cámara nueva. Esta operación se realiza desde la interfaz de mantenimiento de la cámara en http://192.168.1.55/admin/maintenance.shtml

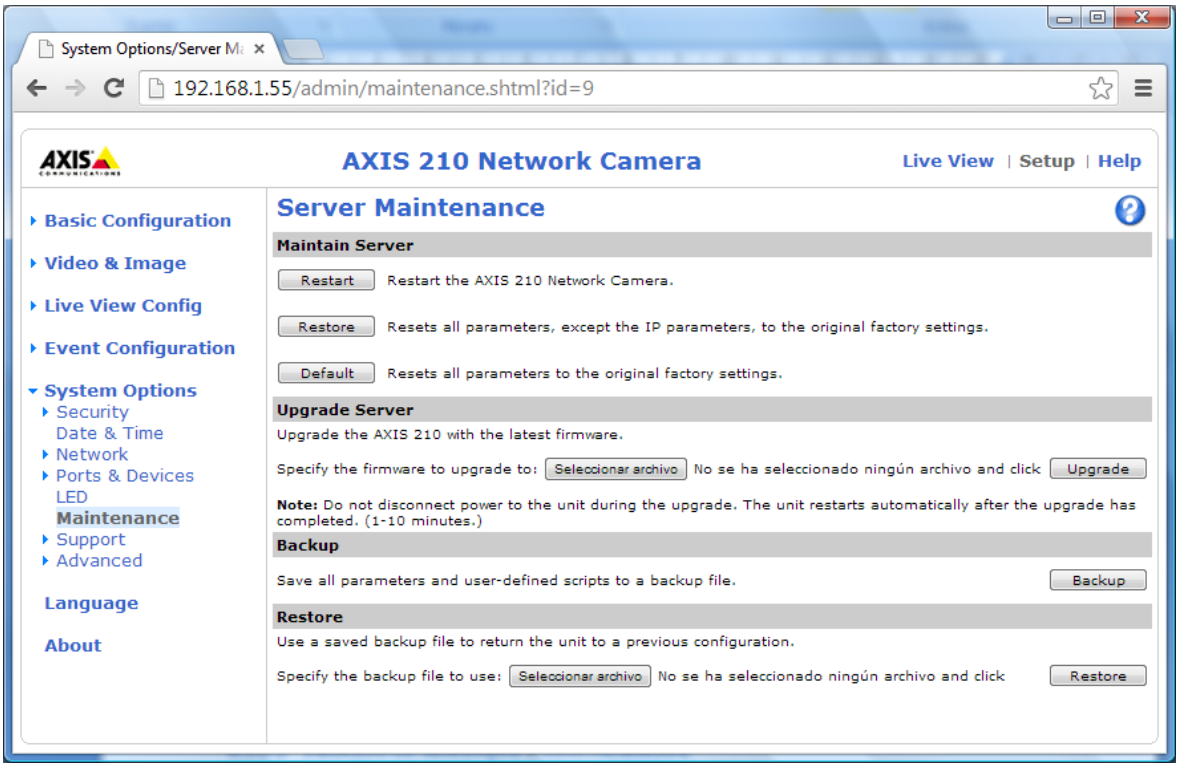


Figura 6.9 Restauración del backup con el software de la cámara

El backup proporcionado contiene todos los scripts, configuraciones y librerías instaladas en la cámara. Es aconsejable realizarlo sobre una cámara nueva para no perder otras configuraciones personalizadas.

Conclusiones:

Como se ha comentado en las conclusiones del capítulo 3, este caso práctico se restringe a entornos productivos con bajas exigencias de velocidad. Hay que recordar que este modelo fue seleccionado por simplicidad y no por su potencia. La cámara a nivel hardware tiene las capacidades suficientes para realizar el modelo planteado cumpliendo todas las expectativas del proyecto y por lo tanto **la idea** de introducir un procesado inteligente en la cámara **resulta ser factible**. Cito al capítulo de introducción:

La idea de este proyecto consiste en incrustar una parte del proceso en la arquitectura de las Cámaras IP dotándolas de cierta inteligencia. Con esto se pretende reducir el ancho de banda consumido y distribuir la carga del proceso.

PRESUPUESTO

A continuación se detalla el presupuesto de un sistema como el del caso práctico, bajo el supuesto de que no hay ninguna infraestructura de red ni de control, que el software cliente lo desarrollarán terceras partes. Quedan excluidas las modificaciones requeridas en la línea de producción.

Concepto	Unid.	Precio u.	Importe
DESPLIEGUE DE MEDIOS			
CÁMARA IP			
Cámara AXIS 210	1	200 €	200 €
CABLEADO DE LA RED			
Switch Ethernet	1	12 €	12 €
Bobina 100m Cable Red Flexible UTP Cat.5e 10/100	1	30 €	30 €
Mano de obra	4	12 €	48 €
CENTRO DE CONTROL			
Equipo de control	1	1.000 €	1.000 €
EXPLOTACIÓN DEL SISTEMA EMPOTRADO EN LAS CÁMARAS IP			
ALGORITMO DE PROCESADO	días / horas	tasa (€/h)	
Toma de requisitos	2 / 16	30	480 €
Diseño	12 / 96	45	4.320 €
Implementación	20 / 160	40	6.400 €
SOFTWARE ADICIONAL			
Servidor DPWS	5 / 40	40	1.600 €
Página web integrada	5 / 40	40	1.600 €
		Subtotal	15.690 €
		IVA 21%	3.295 €
		TOTAL	**18.985 €**

CONCLUSIONES

Independientemente de las dificultades encontradas, la idea de poder dotar a cámaras de vigilancia de cierta inteligencia es muy interesante. Seguramente las cámaras terminen evolucionando para soportar este tipo de aplicaciones. Ya existe el concepto de Smart Cameras y hay soluciones comerciales como la Plataforma de aplicaciones AXIS, en las que terceras partes pueden desarrollar sus algoritmos y comercializarlos fácilmente.

Estaría genial que las aplicaciones de diseño de sistemas como Simulink trabajaran sobre una capa de abstracción del hardware y hubiese podido utilizar el código generado. De hecho estoy seguro que también se podría generar el código para la explotación mediante SOA desde este diseñador de sistemas.

Conectar un dispositivo a la red y poder manipularlo desde cualquier ordenador de forma integrada gracias a DPWS tiene un gran potencial. El poder interconectar varios dispositivos para hacer una aplicación distribuida e inteligente y poder hacer que el mundo que nos rodea nos facilite la vida es un camino muy interesante.

A veces llevar la teoría a la práctica es una barrera infranqueable. Las limitaciones que se pueden encontrar son de lo más variopintas. Por ejemplo, el trabajo con los colores en algoritmos de procesado de imágenes es un mundo lleno de retos y solo es la punta del iceberg en el campo de la visión por computador. Lo más grave es que estos fallos no se detecten hasta encontrarse con un caso apropiado. Como se ha recomendado, para sortear este obstáculo hay que trabajar con una base de imágenes de prueba y otra de contraste.

Es recomendable identificar con qué librería estándar de C se va a trabajar. Respecto a las compilaciones cruzadas es mucho mejor trabajar con procesadores de gran público como ARM, la comunidad de soporte es mucho más grande. Un consejo general en este aspecto es que si algo no compila intentarlo con la versión anterior.

Sobre el procesado de imágenes la recomendación es que no hace falta procesar la imagen al completo, al igual que los humanos no vemos bien en la región periférica las máquinas no tienen porqué hacerlo mejor. Para esto se definen una zonas de interés llamadas ROI – Region of interest, que se pueden definir de forma estática o calcularlas con otro proceso.

GLOSARIO

- **Canales de imagen:** La información de una imagen puede estar dividida o agrupada según determinados parámetros. Los canales más habituales son RGB (Rojo, Verde, Azul), aunque existen varios espacios de color.
- **Conversión a escala de grises:** Proceso que trabaja sobre una imagen en color aplicando una transformación lineal para extraer la luminancia de la imagen, el resultando es una imagen sin información de color llamada blanco y negro o más específico, escala de grises.
- **Imagen en color:** Una imagen en color está compuesta por varias imágenes que representan diferentes parámetros del color.
- **Imagen en escala de grises:** Se la conoce como imagen en blanco y negro, pero siendo más preciso es una imagen que tiene la intensidad de la luz codificada en un amplio rango de valores, generalmente de 0 a 255. Cada canal RGB de una imagen en color son de este tipo.
- **Thresholding:** Proceso que trabaja sobre una imagen en escala de grises asignando el valor 1 a los pixeles de la imagen si superan cierto umbral de intensidad y 0 en caso contrario, resultando en una imagen binaria.
- **Vecindad / Conectados:** Relación entre 2 pixeles por su posición y cercanía. Hay dos casos, vecindad por las aristas (4-conectados) y vecindad por las esquinas (8-conectados). Se dice que dos pixeles están conectados si el pixel evaluado tiene el mismo valor que alguno de sus vecinos.
- **Bloque/Blob:** En una imagen binaria, un conjunto de pixeles conectados con el mismo valor.
- **Llenado:** Proceso que trabaja sobre imágenes binarias de forma que completa los pixeles 1 aislados dentro de un bloque 0 para eliminar el ruido dentro de los bloques de píxeles.
- **Erosión:** Proceso de desgaste de la imagen binaria, elimina pequeñas partículas aisladas.
- **Etiquetado:** Proceso que asigna a cada bloque un identificador.
- **Segmentación**: Proceso que consiste en separar/segmentar los diferentes objetos presentes en una imagen utilizando diferentes características de la imagen.
- **SOA**: Service Oriented Arquitecture. Arquitectura orientada a servicios. Es un paradigma tecnológico para el diseño de aplicaciones basada en servicios. Bajo esta arquitectura las aplicaciones son colecciones estructuradas de servicios individuales.
- **SOA4D:** Arquitectura Orientada a Servicios para Dispositivos es una iniciativa de código abierto con el objetivo de fomentar un ecosistema para el desarrollo de componentes de software orientados a servicios (mensajería SOAP y protocolos WS-*, orquestación de servicios...) adaptado a las limitaciones específicas de dispositivos integrados.

BIBLIOGRAFÍA

1. **AXIS Communications AB.** *IP-Surveillance guide*. [en línea] http://www.axis.com/files/manuals/gd_ipsurv_design_32568_en_0807_lo.pdf
2. **AXIS Communications AB.** *AXIS ETRAX 100LX Designer's Reference* (2006).[en línea] http://www.axis.com/files/tech_overview/etrax_100lx_des_ref-060209.pdf
3. **AXIS Communications AB.** *AXIS Developer Wiki*. [en línea] http://developer.axis.com/wiki/index.html
4. **AXIS Communications AB.** AXIS Developer Wiki. *Software list*. [en línea] http://developer.axis.com/wiki/doku.php?id=axis:sw-list
5. **AXIS Communications AB.** AXIS Developer Wiki. *Compiling for CRIS HOWTO*. [en línea] http://developer.axis.com/wiki/doku.php?id=axis:compiling_for_cris_howto
6. **AXIS Communications AB.** AXIS Developer Wiki. *Applications HOWTO*. [en línea] http://developer.axis.com/wiki/doku.php?id=axis:apps_howto
7. **AXIS Communications AB.** AXIS Developer Web site. *Plataforma de aplicaciones de cámaras AXIS*. [en línea] http://www.axis.com/es/techsup/cam_servers/dev/application_platform/index.htm
8. **David Villa Alises.** *Creación de Librerías en GNU/Linux*. [en línea] http://arco.esi.uclm.es/~david.villa/doc/repo/librerias/librerias.html
9. **Wikimedia Foundation, Inc.** *Comparison of numerical analysis software*. [en línea] http://en.wikipedia.org/wiki/Comparison_of_numerical_analysis_software
10. **M. C. José Jaime Esqueda Elizondo.** *Fundamentos de Procesamiento de Imágenes*. [en línea] http://fcqi.tij.uabc.mx/usuarios/esqueda/cursoimagenes.pdf
11. **The MathWorks, Inc.** *MATLAB Product Help*.
12. **The MathWorks, Inc.** Real-Time Workshop Help. *First Look at Generated Code*. [en línea] http://www.mathworks.com/access/helpdesk/help/toolbox/rtw/gs/bp6h8dw-1.html
13. **ITU-R.** *BT.601-7*. [en línea] http://www.itu.int/rec/R-REC-BT.601-7-201103-I/en
14. **Computer Vision Online.** *Software*. [en línea] http://www.computervisiononline.com/software
15. **Chai Braudo.** *Quick n' Dirty BMP Library*. [en línea] http://qdbmp.sourceforge.net/

16. **Universität Rostock.** *WS4D Reference Manual* (2010). [en línea] http://trac.e-technik.uni-rostock.de/projects/ws4d-gsoap/chrome/site/ws4d-gsoap-refman-0.8.pdf

17. **Microsoft Corporation.** *DPWS Library.* [en línea] http://msdn.microsoft.com/en-us/library/hh423016.aspx

18. **SOA4D.org.** *DPWS Core User Guide*. [en línea] https://forge.soa4d.org/docman/view.php/8/45/DPWSCore+User+Guide.pdf

19. **Phillips, Dwayne.** *Image Processing in C (Second Edition)*. [en línea] 2000 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.177.1793&rep=rep1&type=pdf

20. **Kisacanin, Branislav; Kölsch, Mathias.** *Tutorial on Embedded Computer Vision*. IEEE CVPR 2006. 17 June 2006. Follow-up web page on Embedded Computer Vision. [en línea] http://www.movesinstitute.org/~kolsch/pubs/EmbeddedCV_TutorialCVPR06.html

21. **A. Rowley, Henry.** *Neural Network-Based Face Detection* Carnegie Mellon University; May 1999

22. **Kernel.org.** *Glibc vs uClibc Differences.* [en línea] https://www.kernel.org/pub/linux/libs/uclibc/Glibc_vs_uClibc_Differences.txt